Palestine Polytechnic University

Deanship of Graduate Studies and Scientific Research

Master of Informatics

Arabic Text Generation Using GANs Models

Submitted By
Hana' Qasrawi

Supervised By
Dr. Alaa Halawani

Thesis Submitted in Partial Fulfillment of Requirements of the Degree Master of Science in Informatics

January, 2024

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Arabic Text Generation Using GANs Models**, submitted by **Hana' Qasrawi** in partial fulfillment of the requirements for the degree of Master in Informatics.

Graduate Advisory Committee:

Dr. Alaa Halawani (Supervisor), Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Hashem Tamimi, Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Abualseoud Hanani, Birzeit University.

Signature:_____ Date:_____

Thesis Approved

Dr. Nafez Nasreideen Dean of Graduate Studies and Scientific Research Palestine Polytechnic University

Signature:_____ Date:_____

DECLARATION

I declare that the Master Thesis entitled **"Arabic Text Generation Using GANs Models"** is my original work, and hereby certify that unless stated, all work contained within this thesis is my own independent research and has not been submitted for the award of any other degree at any institution, expect where due acknowledgement is made in the text.

Hana' "Mohammad Rashed" Qasrawi

Signature:_____ Date:_____

STATEMENT OF PERMISSION TO USE

By submitting this thesis for the partial fulfillment of the requirements for the master's degree in Informatics at Palestine Polytechnic University, I hereby consent to its availability for borrowers in accordance with the library's regulations. Limited quotations from this thesis are permissible without specific authorization, provided proper acknowledgment of the source is included. For extensive quoting, reproduction, or publication of this thesis, permission may be sought from my primary supervisor. In the absence of the supervisor, such permission can be granted by the Dean of Graduate Studies and Scientific Research, contingent upon their judgment that the intended use is for scholarly purposes. Unauthorized reproduction or use of the material in this thesis for financial gain is strictly prohibited without obtaining my written consent.

Hana' "Mohammad Rashed" Qasrawi

Signature:_____ Date:_____

الملخص

إن إنشاء النصوص يعد واحداً من أهم المهام في ميدان التعلم العميق ومعالجة اللغات الطبيعية، حيث تهدف لخلق مخرجات جديدة استناداً إلى سلاسل من النصوص أو الصور المدخلة، ويتيح هذا النهج تحسين العديد من التطبيقات في مجال الروبوتات، والتقنيات المساعدة، وسرد القصص، وفنون التأليف، ومجالات أخرى.

تستخدم نماذج التعلم العميق لتحليل مجموعات ضخمة من البيانات، سواء كانت نصية أو صورية، بهدف خلق كلمات أو جمل بتنسيقات متنوعة. ومن بين هذه النماذج، نجد شبكة الخصومة التوليدية (GAN)، حيث يتم تنفيذها كنظام يتألف من شبكتين عصبيتين هما: شبكة التمييز وشبكة التوليد. يتولى الجزء الأول مسؤولية تحديد ما إذا كانت البيانات مزيفة أم حقيقية، بينما يقوم الجزء الثاني بإنتاج بيانات جديدة.

في إطار هذه الأطروحة، سنعتمد على نماذج GAN محددة وهي (SeqGAN, TextGAN, RankGAN) لتدريبها على بيانات نصية باللغة العربية، بهدف إنشاء جمل صحيحة ومترابطة. سنقوم أيضاً بتقييم النصوص الناتجة من هذه النماذج والمقارنة بينها للحصول على أفضل النصوص باللغة العربية، ثم نقوم بمقارنة شاملة بين النتائج لهذه النماذج سواء باللغة العربية أو الإنجليزية.

Abstract

Text generation, a fundamental task in Deep Learning and Natural Language Processing, involves crafting new outputs based on inputted text or images, with far-reaching applications in robotics, assistive technologies, storytelling, and more. The Generative Adversarial Network (GAN), a dual-neural network system, engages in a zero-sum competition where a discriminative network assesses the authenticity of generated data from a generative network. This adversarial training encourages the discriminator to maximize the probability of accurately classifying both real and generated data, while the generator seeks to minimize it, establishing a dynamic equilibrium.

Recent advancements in GANs models highlight their efficacy in English text generation. By training on a text dataset, the discriminator learns intricate word and sentence formulations, empowering the generator to produce text in the desired language. This thesis takes a novel approach, applying SeqGAN, TextGAN, and RankGAN models to an Arabic dataset extracted from artistic and cultural news articles, meticulously split into thousands of sentences. The use of TextBox tools, specifically tailored for GANs models, is instrumental in fine-tuning these models for accurate Arabic sentence generation. Rigorous evaluations of each GAN model's output are conducted, paving the way for a comprehensive comparative analysis, for offering nuanced insights and recommendations into their performance.

In honor of the resilient people of Palestine, the valiant resistors, the righteous martyrs, and the great Tufan Al-Aqsa, this dedication is made to the wounded and proud land.

Acknowledgements

Foremost, I extend my heartfelt gratitude to Allah, who gave me the strength and the ability to complete this thesis. Without His divine help, this achievement would have remained beyond my reach.

Indeed, numerous individuals have played pivotal roles in bringing this dissertation to fruition, and to them, I owe immense gratitude. My deepest appreciation goes to my supervisor, Dr. Alaa Halawani, whose unwavering support and guidance have been invaluable on this academic odyssey.

I reserve my sincerest thanks for my cherished parents and family, whose unwavering support and encouragement, especially during challenging moments, were the bedrock upon which the entire thesis endeavor was built. Their constant presence made the journey meaningful and the accomplishment truly significant.

Special thanks are due to my grandmother, whose prayers for my success and excellence added an extra layer of blessings to this endeavor. Her silent support has been a source of inspiration and strength.

To all of you who have contributed to this journey in various ways, my heartfelt appreciation. Grateful to all of you for being part of this significant chapter in my academic and personal growth.

Contents

List of Figures

List of Tables

Chapter One

Introduction

A language model (LM) is a key component in Natural Language Processing (NLP) and is designed to understand and generate human language. The concept of language modeling has been integral to various language-related tasks, such as machine translation, speech recognition, text summarization, sentiment analysis, and text generation. The fundamental goal of a LM is to predict the probability distribution of words in a given sequence of text.

Early LMs were based on n-grams, which are sequences of n consecutive words. These models, although simple, suffered from limited context understanding and struggled to capture long-range dependencies in language.

With the emergence of deep learning, particularly neural networks, LMs witnessed significant advancements. The transformer architecture, introduced in 2017, revolutionized language modeling. The Transformer's self-attention mechanism enabled the model to effectively weigh the importance of each word in the input sequence based on its context, capturing complex linguistic relationships.

Pre-training became a pivotal aspect of modern LMs, allowing them to leverage vast amounts of unlabeled text data for initial learning. Pre-trained LMs, like OpenAI's GPT (Generative Pre-trained Transformer), have demonstrated impressive performance across various NLP tasks, thanks to their ability to generalize knowledge from pre-training to downstream tasks.

The Arabic language is one of the most widely spoken languages globally, with rich cultural and historical significance. Developing LMs specifically for Arabic poses unique challenges due to the language's distinctive characteristics, including its right-to-left script, complex morphology, and variations in dialects.

Initially, linguistic models for Arabic faced limitations due to the lack of Arabic training data and the complexity of Arabic text processing. However, with the rise of deep learning and the success of Transformers construction, Arabic language modeling has improved dramatically.

Arabic LMs utilize sub-word tokenization techniques, like Byte-Pair Encoding (BPE) or Sentence-Piece, to handle the vast vocabulary and word morphology. Additionally, pre-training on large-scale Arabic corpora allows models to learn the underlying linguistic patterns and structures in Arabic text.

Fine-tuning these pre-trained models on task-specific data, such as poetry generation, machine translation, or sentiment analysis, enhances their performance on specific Arabic language generation tasks. Moreover, advancements in transfer learning techniques have enabled knowledge transfer from pre-trained models to tasks with limited data, making it possible to develop effective Arabic LMs even in low-resource scenarios.

As research in NLP for Arabic continues to progress, LMs for the Arabic language are continuously evolving, opening up new possibilities for enhanced natural language understanding and generation for Arabic-speaking communities.

Using TextBox [1], we will train GANs models on Arabic datasets to generate accurate and contextually appropriate Arabic sentences. TextBox offers transformer-based architectures like GPT and BERT for language modeling. The GANs models consist of a generator and a discriminator network engaged in competitive learning. Through this research, we aim to advance Arabic text generation and explore its applications. Evaluating the models with the Bleu-n metric will provide insights into their accuracy and fluency. Leveraging TextBox's NLP tools, we enhance GANs' Arabic language processing capabilities.

1.1 Main Contributions

This thesis makes significant contributions to the field of NLP, particularly in the area of Arabic text generation using Generative Adversarial Networks (GANs). The primary contributions are outlined below:

- **Development and Training of GAN Models for Arabic Text Generation:**

  This research focuses on training GAN models, specifically SeqGAN, TextGAN, and RankGAN, using Arabic datasets to generate accurate and contextually appropriate Arabic sentences. By leveraging the open-source library TextBox, which provides transformer-based architectures like GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers), the study aims to advance the capabilities of GANs in handling the unique characteristics of the Arabic language.

- **Majoring Use of SeqGAN, TextGAN, and RankGAN for Arabic Text Generation:**

  This work is the first to address the use of SeqGAN, TextGAN, and RankGAN models in generating texts in Arabic. Each of these models brings unique strengths to the table: SeqGAN focuses on sequence generation, TextGAN on text quality, and RankGAN on ranking and optimizing generated text based on quality metrics.

- **Integration of Transformer-based Architectures:**

  TextBox's transformer-based architectures are utilized to enhance the GAN models' performance in understanding and generating Arabic text. The self-attention mechanism of transformers allows the models to effectively weigh the importance of each word in the input sequence based on its context, thereby capturing complex linguistic relationships inherent in the Arabic language.

- **Comprehensive Evaluation Using BLEU-n Metrics:**

  To assess the performance of the trained GAN models, the BLEU-n metric is employed. This evaluation method measures the similarity between the generated text and reference text from the datasets, providing insights into the models' accuracy and fluency. By analyzing the BLEU scores, the research evaluates how well the models capture the linguistic nuances of Arabic.

- **Advancements in Arabic NLP:**

  The research aims to contribute to the advancement of Arabic NLP by exploring the potential applications of the generated text in various domains, including creative writing, dialogue generation, and story summarization. This exploration highlights the practical implications of the study and its potential to enhance further developments in Arabic language processing.

- **Detailed Parameter Optimization and Model Configuration:**

  Extensive experimentation and fine-tuning of model parameters were conducted to identify the optimal configurations for the GAN models. This includes adjusting the generator and discriminator embedding sizes, hidden layer dimensions, dropout rates, and Reinforcement Learning (RL) parameters. These systematic tests were contributory in deciding the configurations that yield the highest quality Arabic sentences.

Through these contributions, the thesis not only advances the technical capabilities of GANs for Arabic text generation but also provides valuable insights and methodologies that can be applied to other low-resource languages. The integration of advanced transformer-based architectures and the thorough evaluation process underscore the innovative approach taken in this research. This majoring work sets a foundation for future studies to build upon and explore the vast potential of GANs in generating Arabic text.

1.2 Thesis Structure

This thesis is composed of six chapters. Chapter 1 gives an introduction, discusses the problem statement, and explains the main contribution in this thesis. In Chapter 2, we delve into the background and literature, exploring NLP, machine learning concepts, and related studies in Arabic text generation. Chapter 3 focuses on Text Generation using GANs, with an emphasis on RL principles and the architecture of SeqGANs tailored for sequential data like text.

Chapter 4 provides a description of the methodology, and details the critical roles of the analysis framework, computer environment, and TextBox tools. It analyzes the structure of the SeqGAN, TextGAN, and RankGAN models, and reveals hyperparameters. Chapter 5 presents experimental results investigating the results of SeqGAN, TextGAN, and RankGAN models for Arabic text generation. The study not only investigates these results but also draws on comparisons with previous studies on Arabic text generation and extends them to a comparative analysis with English results using a GANs model. Finally, Chapter 6 presents the conclusion and future work.

Chapter Two

Background and Literature Review

In this chapter, our focus will be on NLP in various domains, including language models, n-grams, followed by a deeper explanation of some of the machine learning concepts that support the scope of our work. In addition, we will discuss some of the related studies achieved on texts generation task and explain other related works.

2.1 Natural Language Processing

NLP is a dynamic research field that revolves around the interaction between computers and human language. Its primary objective is to develop advanced algorithms and models capable of understanding, interpreting, and generating human language in a meaningful and useful manner. NLP encompasses a wide spectrum of tasks, ranging from fundamental text classification and sentiment analysis to complex tasks like machine translation, question answering, and text generation.

One pivotal area of NLP is natural language text processing, which involves manipulating texts for knowledge extraction, automatic indexing, abstracting, and formatting. The goal is to structure vast amounts of textual information, enabling efficient retrieval of specific data and derivation of knowledge structures tailored to specific purposes. Automatic text processing systems play a central role in transforming text inputs into diverse output forms, enhancing the accessibility and usability of textual data.

At the core of natural language text processing lies the translation of potentially ambiguous natural language queries and texts into unambiguous internal representations. These representations form the basis for matching and retrieval, allowing the system to accurately respond to user queries and extract relevant information.

The text processing pipeline typically begins with morphological analyses, wherein terms in both queries and documents undergo stemming to obtain their morphological variants. This process enhances matching and retrieval capabilities by effectively handling different word forms.

Furthermore, lexical and syntactic processing steps involve utilizing lexicons to determine word characteristics, including meanings, grammatical categories, and relationships with other words. By employing parts-of-speech recognition and sentence parsing, the system can identify

relevant words and phrases and analyze the syntactic structure of sentences, improving contextual understanding and meaning extraction [2].

Past research in natural language text processing has been extensively reviewed by esteemed scholars like Haas (1986), Mani & Maybury (1999), Smeaton (1999), and Warner (1987) [2]. Notably, some NLP systems specialize in processing texts using specific sublanguages, reducing computational complexity and catering to specific complexities. These domain-specific studies, known as 'sublanguage analyses,' have focused on various subject areas, such as medical science or patent texts, aiming to enhance the accuracy and efficiency of text processing for specific domains [2].

In summary, NLP is a diverse and vibrant field that delves into the intricacies of human language. Natural language text processing, an integral part of NLP, seeks to manipulate and structure texts for knowledge extraction and effective information retrieval. By leveraging advanced techniques and models, NLP continues to drive innovation in various applications, including text generation and comprehension, propelling us toward more sophisticated and interactive human-computer interactions.

2.1.1 Language Modeling

A LM is a fundamental component in NLP that plays a key role in understanding and generating human language. It has become integral to various language-related tasks, including machine translation, speech recognition, text summarization, sentiment analysis, and text generation. The primary objective of a LM is to predict the probability distribution of words in a given sequence of text.

LM needs to learn natural language grammar, syntax, and semantic, it is used as root architectures for larger NLP tasks, such as speech recognition, spelling correction, machine translation, and a trained LM can generate new text according to the structure it has already learned. There are primarily two types of LM:

- **Statistical LMs:** These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMMs), and certain linguistic rules to learn the probability distribution of words [3].
- **Neural LMs:** They use different kinds of Neural Networks to model language like (RNN, LSTM, and GAN), and have surpassed the statistical LMs in their effectiveness [3].

Statistical LMs have a strong binding force on the appearance of the current word but require a considerable amount of training text to determine the parameters of the model. When the data set is large, the parameter space of the model will be sparse which causes a data smoothing problem. In addition, these models have some limitations such as being constructed based on discrete unit words, thus do not have the semantic advantages satisfied by word vectors in Neural LMs, such as words of similar meaning have similar word vectors which are used as a system [3].

Neural LMs offer rich representability and information in the data, the model is completely automatically constructed from a parallel corpus and usually, no human intervention is needed. The representations of data in different forms, such as text and image, can all be learned as real-valued vectors which makes it possible to perform information processing across multiple modalities [4].

By predicting word probabilities in a sequence, LMs enable the generation of coherent and contextually appropriate text [3]. The advent of the Transformer architecture, incorporating self-attention and pre-training techniques, has brought about a revolution in language modeling [5]. Pre-trained LMs like BERT and GPT have emerged as prominent examples, significantly advancing the performance of NLP tasks.

Traditional LMs, such as n-gram models, laid the foundation for understanding sequences of words, but they faced limitations in capturing long-range dependencies and context understanding. The advent of deep learning and neural networks introduced groundbreaking models like the Transformer architecture. This architecture, introduced in the seminal [6], revolutionized language modeling and became a cornerstone for diverse NLP tasks.

LMs play a role in the advancement of AI and human-computer interactions. They are constantly evolving, driving research and development NLP. These models contribute to advancements in both understanding and generating human text.

2.1.2 N-gram

Early LMs were based on N-grams, which are sequences of N consecutive words in a text. These models used a straightforward approach to language analysis by considering the frequency and distribution of word sequences in the data. While N-grams provided a simple way to represent language, they had limitations in capturing context and long-range dependencies [5].

The term "N" in N-gram represents the number of elements or units considered together in a sequence. For example:

- A unigram (1-gram) represents a single word or character.
- A bigram (2-gram) represents a sequence of two consecutive words or characters.
- A trigram (3-gram) represents a sequence of three consecutive words or characters.
- And so on, such as 4-gram, 5-gram, etc.

N grams play a role, in capturing the context and statistical patterns within data. They provide insights into word or character frequency and distribution within a given text or collection of texts [5].

In Arabic language modeling N grams are utilized to estimate the probability of a word based on the preceding words in a sequence. This enables the LM to comprehend the context and dependencies in Arabic text.

To illustrate this lets take the sentence ".أحب اللغة العربية" in this sentence the 2-grams would be:

- "أحب اللغة"
- "اللغة العربية"

A 2-grams LM for Arabic would learn how likely it is to encounter words, like "اللغة" when preceded by "أحب" and similarly how probable it is to find "العربية" after "اللغة." This understanding allows for generating text with context during text generation.

While N-grams serve as a fundamental representation for text generation and analysis, they may now not be fully adequate for producing coherent and contextually appropriate text in Arabic, particularly for longer sequences [7]. This limitation arises from fixed window size of N words, which restricts N-gram models to consider only a local context around each word. As a consequence, these models face challenges in capturing difficult linguistic relationships and dependencies [8].

To overcome these limitations, researchers have developed more advanced text generation models, such as neural LMs. One notable example is the Transformer-based architecture, which has demonstrated remarkable capabilities in capturing long-range dependencies in Arabic text. By leveraging a self-attention mechanism, these models can dynamically weigh the significance of each word in the entire sequence, providing a comprehensive contextual understanding of the details of the language.

For instance, a neural LM can take the Arabic input "أحب" (meaning "I love") and generate contextually appropriate text, such as "أحب القراءة والكتب" (meaning "I love reading and books"), based on its profound understanding of the global context and dependencies in Arabic. This ability to generate coherent and relevant text showcases the effectiveness of neural LMs in capturing the intricacies of the Arabic language and producing high-quality output.

2.2 Machine Learning

In the domain of NLP, statistical and machine learning techniques play a pivotal role in developing algorithms that enable computer programs to understand and manipulate human language. These algorithms aim to identify patterns and relationships within example or "training" data, which then allows the program to generalize its knowledge and make predictions about new data [9].

The learning process involves computing numerical parameters that characterize the underlying model of a given algorithm. This computation is achieved through iterative optimization of a numerical measure, fine-tuning the model to perform better on the task at hand.

Supervised learning and unsupervised learning are two common approaches in NLP. In supervised learning, each item in the training data is provided with a corresponding label or correct answer. This labeled data is used to teach the algorithm to recognize patterns and make accurate predictions. On the other hand, unsupervised learning operates on unannotated data and aims to

identify patterns and structures without explicit guidance. Techniques such as cluster and factor analysis fall into the category of unsupervised learning [10].

An inherent challenge in any learning approach is the risk of overfitting. Overfitting occurs when a model becomes excessively tailored to fit the training data but performs poorly on new, unseen data. This issue arises when the model learns not only the essential features but also the random noise present in the training data. To address this concern, researchers use techniques like cross-validation, where the example data is randomly partitioned into training and test sets. By internally validating the model's predictions through multiple rounds of partitioning, training, and validation, the risk of overfitting can be mitigated [11].

Machine-learning models in NLP can be broadly categorized as generative or discriminative. Generative methods focus on creating rich models of probability distributions. Such models can "generate" synthetic data, which means they can simulate the distribution of data from a given source. In contrast, discriminative methods are more utilitarian and directly estimate posterior probabilities based on observed data. An analogy to illustrate the difference between the two is that generative approaches draw from deep knowledge of numerous languages to perform tasks, while discriminative methods rely on differences between languages to find the best match [11].

Generative models can become computationally challenging when dealing with a large number of features. Discriminative models, on the other hand, typically handle more features and are more flexible in adapting to complex data. Examples of discriminative methods include logistic regression and conditional random fields (CRFs), while examples of generative methods include Naive Bayes classifiers and HMMs.

These machine-learning methods have profound implications for NLP tasks and applications. They contribute to the advancement of NLP research and enable the development of sophisticated language processing systems that can understand, generate, and manipulate human language in meaningful and useful ways.

2.3 Text Generation Task

Text generation is a fascinating task within NLP, where the goal is to create text that is indistinguishable from human-written text. To achieve text generation, a LM undergoes a crucial training process where it learns a probabilistic distribution that accurately represents the patterns and structures in the training data. Once trained, the LM can then sample new data based on this learned distribution [12].

In the context of text generation, the LM estimates the probability distribution of a token at a specific time step, given a sequence of previous tokens. This sequence can be of length n, representing the n-sized context of previous tokens. By estimating the probabilities of a token w_t based on the preceding sequence of tokens $P(w_t|w_{t-1},w_{t-2},..., w_{t-n})$, the LM can generate a new token at the current time step t [12].

The text generation process proceeds iteratively, as the LM generates one token at a time and moves on to the next time step $t + 1$. This sequential generation continues until a stopping

criterion is met, often defined by the number of tokens to be generated or the achievement of convergence [12].

After generating a sequence of tokens, the resulting tokens are arranged sequentially to form the final generated text. The LM's ability to accurately estimate the probabilities of tokens based on contextual information allows it to produce coherent and contextually appropriate text that resembles human-written language.

Text generation is a challenging yet crucial task in NLP, finding applications in various domains such as creative writing, chatbots, content generation, and more. Advanced LMs, like the Transformer-based architectures with self-attention mechanisms, have shown impressive capabilities in text generation, pushing the boundaries of what is possible in natural language generation [13].

2.4 Embedding Systems

In NLP, embedding systems play a definitive role in text generation tasks. Text generation involves producing coherent and contextually relevant sentences, paragraphs, or even longer pieces of text based on a given input or context. Embeddings are essential for this task as they provide a way to encode semantic meaning and contextual information into dense vector representations, which models can then use to generate text [14]. Role of Embeddings in Text Generation:

- **Semantic Representation:** Embeddings represent words, phrases, or sentences as continuous vectors in a high-dimensional space. These vectors capture semantic relationships, meaning that words with similar meanings are placed close to each other in this space. This allows the text generation model to understand and use these semantic relationships to generate meaningful text.
- **Contextual Understanding:** By using embeddings, models can capture the context in which words appear. Contextual embeddings, such as those from models like BERT or GPT, dynamically generate embeddings based on the surrounding text, providing a nuanced understanding that improves the coherence and relevance of generated text.
- **Sequence Modeling:** In text generation, maintaining the sequence and structure of the text is critical. Embeddings help models to remember and generate sequences by providing a continuous representation that captures the relationships between different parts of the text.

Types of Embeddings Used in Text Generation:

- **Word Embeddings:** Techniques like Word2Vec, GloVe, and FastText are used to create word embeddings that capture the meanings of individual words. These embeddings serve as the building blocks for generating text by providing a basic semantic framework.
- **Subword Embeddings:** Subword embeddings, such as those used in BERT and GPT, break down words into smaller units, capturing meaning even for rare or unseen words.

This approach helps in generating text that is both linguistically diverse and syntactically correct.

- **Sentence Embeddings:** Sentence embeddings represent entire sentences as single vectors. Models like Sentence-BERT provide embeddings that capture the overall meaning and intent of a sentence, enabling the generation of coherent and contextually appropriate responses.

In GAN models, embeddings serve as a fundamental component for representing input data effectively, particularly in text generation tasks. These embeddings play a necessary role in encoding the semantic meaning of words and their contextual relationships within the text corpus.

In the context of GANs for text generation, the generator network typically takes random noise vectors or latent variables as input, along with additional information such as class labels or conditioning variables. These input vectors are then transformed into text embeddings using pre-trained word embeddings or learned embeddings within the model.

During the training process, the generator network learns to generate text samples by manipulating the input embeddings in the latent space. The embeddings undergo transformation through several layers of the generator network, ultimately producing text samples that closely resemble the training data.

Meanwhile, the discriminator network in the GAN model evaluates the authenticity of generated text samples by comparing them to real text samples from the training data. This assessment may also involve operating on embeddings to evaluate the semantic coherence and relevance of the generated text.

Once trained, the generator network can produce new text samples by transforming random noise vectors or latent variables into text embeddings and then decoding these embeddings into coherent text sequences using techniques like sampling or beam search.

In SeqGAN, TextGAN, and RankGAN, the embedding system plays an important role in representing and processing text data for effective generation.

In SeqGAN, the embedding system typically employs pre-trained word embeddings, such as Word2Vec or GloVe, to represent individual words as dense vectors. These word embeddings are used to encode the input text data, enabling the generator network to understand the semantic similarities between words and generate coherent text sequences. During training, the generator network learns to manipulate these embeddings in the latent space to produce text samples that closely resemble the training data.

Similarly, TextGAN utilizes embeddings to represent words and encode textual information. The generator network in TextGAN transforms random noise vectors or latent variables into text embeddings, which are then decoded into text sequences. The discriminator network evaluates the authenticity of generated text samples by comparing their embeddings to those of real text samples from the training data. By operating on embeddings, TextGAN effectively captures semantic coherence and relevance in the generated text.

In RankGAN, the embedding system serves a similar purpose in encoding textual information for generation. The generator and discriminator networks in RankGAN interact with embeddings to produce and evaluate text samples, respectively. These embeddings enable RankGAN to understand the semantic context of words and generate coherent text sequences that involve to the linguistic patterns of the training data.

2.5. Related Work

In recent years, the field of NLP has experienced in interest, particularly in the context of Arabic text generation as sentences or poems, similar to English and Chinese languages. The exploration of various NLP domains, such as machine translation, information retrieval, and text summarization, has highlighted the critical need for robust Arabic language resources. Given the abundance of information in Arabic on the web, the accurate interpretation of this content, particularly textual data, has become increasingly essential. In this thesis, we delve into the field of generating accurate Arabic sentences using GANs models. Here, we present some of various researches focused on text generation in Arabic, English, and Chinese. We will later use these studies to compare their outcomes with our own results.

Souri et al. [15] employed Recurrent Neural Networks (RNNs) LMs on the Arabic language, conducting training and testing on free Arabic text datasets from "Arab World Books" and "Hindawi." Given that the conventional RNN architecture does not seamlessly align with Arabic language nuances, the researchers tailored an RNN model to accommodate Arabic features, introducing a gated Long-Short Term Memory (LSTM) model designed to address specific Arabic language criteria. This work marked the initiation of applying deep learning models to Arabic text. It demonstrated that LSTM models can be applied to Arabic with some modifications to account for Arabic language characteristics. In their experiments, standard LSTM architecture was first applied to Arabic, yielding a loss value of 1.43, which was then compared to the gated LSTM model with a reduced loss value of 0.73, showcasing enhanced accuracy. Additionally, the standard LSTM was applied to Arabic, English, and Chinese to observe model behavior across different languages, resulting in a loss value of 1.43 for Arabic, 2.13 for Chinese, and 1.2 for English.

Talafha et al. [16] was the first to propose an Arabic poem generation model using deep learning algorithms. The first part is a Bi-directional Gated Recurrent Unit (Bi-GRU) to generate the first line. The second part is a modified Bi-GRU encoder-decode. The proposed approach uses a hybrid model that combines a TextRank algorithm and a word embedding technique to extract keywords. FastText approach used to build the word-level embedded model. The dataset contains 80,506 verses from 20,106 Arabic poems that expressed love and religion. Quantitative evaluation using BLEU scores shows that the proposed model outperforms other deep learning approaches. In terms of the BLEU-1 score (0.412), BLEU-2 score (0.314), BLEU-3 score (0.204) and BLEU-4 score (0.109).

In a more recent work, Talafha et al. [17] propose a poetry generation model with enhanced phonetic and semantic embeddings. In this work, they propose a three phased approach. First, they

use a word-embedding model that represents verses' rhyme and rhythm besides the context. These embeddings offer information on each word's phonetics and its vectorized word representation. Second, they extract N keywords representing the sub-themes of the N verses to generate, where each keyword is to generate one new verse. In the third phase, they use a Bi-GRU encoder-decoder model with word-level and verse-level attention to generate the first verse. Then, they use a hierarchical sequence-to-sequence model to produce the next verses. This last work approach assumes that the number of the extracted keywords is equal to the number of verses to generate. If the input is too short, it will be impossible to generate enough verses. Quantitative evaluation using BLEU scores shows that the proposed model outperforms other deep learning approaches. In terms of the BLEU-1 score (0.53), BLEU-2 score (0.4), BLEU-3 score (0.3) and BLEU-4 score (0.15).

Recently, GANs [18] have been widely explored due to its nature of unsupervised deep learning. While GANs have demonstrated remarkable success in computer vision applications, their application to NLP has been limited due to the non-differentiability of discrete sequences. To overcome this limitation Yu et al. [19] proposed SeqGAN model that presents a solution by incorporating policy gradients inspired by RL Long Short-Term Memory (LSTM). Here, each word selection in a sentence is considered an action, and the sequence's reward is computed using Monte Carlo (MC) search. The method involves back-propagating the reward from the discriminator, steering the generator toward producing language sentences that closely resemble human expression.

Yu et al. [19] proposed SeqGAN to generate Chinese poems and Barack Obama political speeches. SeqGAN performance in generating creative sequences, as demonstrated by its BLEU-2 score of 0.74 for Chinese poem generation and BLEU-4 score of 0.43 for Obama political speech generation. These results underscore SeqGAN's effectiveness in generating diverse and contextually relevant language sequences.

Zhang et al. [20], the authors introduce TextGAN, aimed to reduce challenges in generating realistic sentences through GANs. Their approach involves an LSTM [21] RNN as the generator and a Convolutional Neural Network (CNN) [22] as the discriminator. The strategy supports the model to learn representations that are both informative about the original sentences (via the autoencoder) and discriminative toward synthetic sentences (via the discriminator). Additionally, the authors propose extra techniques, including initialization strategies and discretization approximations, to facilitate GAN training, achieving superior performance compared to related approaches. Experimental validation was conducted using 0.5 million sentences from BookCorpus and arXiv for training and validation, with 25,000 randomly selected sentences from both corpora for testing. Evaluation included the generator's loss and corpus-level BLEU score. A comparative analysis with seqGAN showcased TextGAN's higher scores, with BLEU-4 at 0.11, BLEU-3 at 0.52, and BLEU-2 at 0.85.

Lin et al. [23], they propose a GAN, RankGAN, for generating highquality language descriptions. RankGAN is able to analyze and rank a collection of human-written and machine written sentences by giving a reference group. By viewing a set of data samples collectively and evaluating their quality through relative ranking scores, the discriminator is able to make better

assessment which in turn helps to learn a better generator. The proposed RankGAN is optimized through the policy gradient technique. They test their method on the image captions provided by the COCO dataset, consisting of caption sentences generated by humans. The training set comprises 80,000 randomly selected captions, and a validation set is formed with 5,000 captions. Notably, RankGAN achieves superior scores with BLEU-2 at 0.845, BLEU-3 at 0.668, and BLEU-4 at 0.614 compared to SeqGAN, consistent with findings in Chinese poem composition.

Chapter Three

Text Generation Using Generative Adversarial Networks

In this chapter, we delve into a comprehensive exploration of components linked to RL is a paradigm within machine learning where an agent learns to make decisions by interacting with an environment. It involves an agent, a set of actions it can take, an environment, and a feedback mechanism in the form of rewards or penalties.

Furthermore, we explain the computational techniques for two classifications of policy gradient. Subsequently, our focus shifts to the architecture of GANs are a class of machine learning models designed for unsupervised learning. They consist of two neural networks, a generator, and a discriminator, engaged in a competitive process to improve the generation of data.

Moreover, we focus into Sequence Generative Adversarial Nets (SeqGANs), a specific variant tailored for sequential data, such as text generation. This section involves a thorough examination of the complexity involved in training SeqGANs to generate coherent and contextually relevant sequences, with a particular emphasis on text generation.

3.1 Introduction to Reinforcement Learning

RL is a prominent branch of Artificial Intelligence that focuses on developing agents capable of making intelligent decisions in dynamic and uncertain environments. Unlike other machine learning paradigms, RL takes inspiration from behavioral psychology, where agents learn to take actions to maximize cumulative rewards based on feedback from the environment [24].

The RL framework revolves around the concept of an agent interacting with an environment over multiple time steps. At each step, the agent observes the environment's state and selects an action based on its current policy. The environment responds to the agent's action by transitioning to a new state and providing a reward signal, indicating the goodness or undesirability of the action taken [24].

The primary goal of RL is for the agent to learn an optimal policy that maximizes the expected cumulative reward over time. This requires striking a balance between exploration and exploitation, as the agent must explore new actions to discover the most rewarding strategies while exploiting already known good actions [24]. RL has five main elements: an environment, an agent,

a policy, a reward signal, and a value function. Figure 3.1 shows how these basic elements interact with each other.

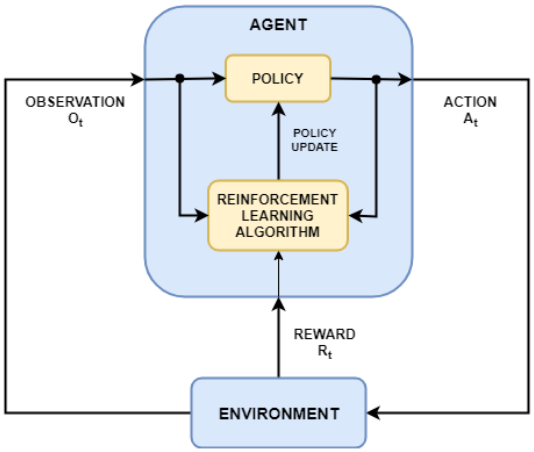

Figure 3.1: RL basic elements [25]

- **Environment:** The environment represents the external system with which the RL agent interacts. It can be real or simulated and provides the agent with observations, representing the current state of the environment. The agent's actions influence the environment, and the environment, in turn, generates new observations and rewards based on the agent's actions.
- **Agent:** The agent is the learner in the RL system. It is the decision-maker that takes actions based on the observations it receives from the environment. The goal of the agent is to learn a policy that maps observations to actions, maximizing the cumulative rewards it receives over time.
- **Policy:** The policy is the core component of an RL agent's decision-making process. It is a strategy that defines the agent's behavior, determining the action the agent should take in a given state of the environment. The policy can be deterministic, where it directly maps states to actions, or stochastic, where it provides a probability distribution over actions for each state. In the next section, we will offer a more comprehensive explanation of the policy, delving into its complexities and the underlying mechanisms, and how the policy influences the generation of synthetic data in the context of GANs.
- **Reward:** The reward is a scalar signal that the environment provides to the agent after each action. It serves as feedback to the agent, indicating the immediate desirability of its action in a particular state. The agent's objective is to learn a policy that maximizes the expected cumulative reward over time.
- **Value Function:** The value function is used to estimate the expected cumulative reward an agent can achieve from a given state or state-action pair. It is a crucial component in RL algorithms and helps the agent evaluate the desirability of different states. There are two main types of value functions: state-value function (V-value function), which estimates the

expected cumulative reward from a given state, and action-value function (Q-value function), which estimates the expected cumulative reward from taking a specific action in a given state.

RL agents learn by iteratively interacting with the environment, observing states, taking actions, receiving rewards, and updating their policies based on the observed outcomes. This learning process, often guided by various RL algorithms, enables the agent to improve its decision-making capabilities and ultimately find strategies that lead to higher cumulative rewards in the given environment. RL has been successfully applied in a wide range of applications, including game playing, robotics, autonomous systems, recommendation systems, finance, and more.

3.2 Policy Gradient

In RL, policy search [26] [27] is divided into two categories: model-free policy search and model-based policy search. Model-free policy search can further be classified as stochastic policy search and deterministic policy search. Stochastic policy gradient (SPG) belongs to the category of basic stochastic policy search, which considers both the state and action spaces. Referring to [26] that explains the calculation method:

Given a set of state-action trajectories τ:$\{s_0, a_0, \cdots, s_T, a_T\}$, where T represents the number of state-action pairs in a trajectory, the total reward of the trajectory τ is defined as:

$$R(\tau) = \sum_{t=0}^{T} r(s_t, a_t) \tag{3.1}$$

Where $r(s_t, a_t)$ represents the immediate reward at state s_t and action a_t. In SPG, a θ-parameterized stochastic policy denoted as π_θ, and the probability of following trajectory τ under policy π_θ can be expressed as:

$$P(\tau;\theta) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t)\, P(s_{t+1}|s_t) \tag{3.2}$$

Where $\pi_\theta(a_t|s_t)$ is the probability that the agent selects action a_t at state s_t under policy π_θ, and $P(s_{t+1}|s_t)$ is the transition probability from state s_t to state s_{t+1}.

The goal is to determine the optimal parameter θ that maximizes the expectation of cumulative rewards:

$$J(\theta) = E_{\pi_\theta}[R(\tau)] = \sum_{\tau} P(\tau;\theta)\, R(\tau) \tag{3.3}$$

Where the expectation is taken over all possible trajectories τ. The gradient of the objective function with respect to θ can be calculated as:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta)\,\nabla_\theta \log[P(\tau;\theta)]R(\tau) \tag{3.4}$$

Where $\nabla_\theta \log[P(\tau;\theta)]$ represents the gradient of the log probability of trajectory τ under policy π_θ. The steepest-descent method with the learning rate η can be used to update the old parameter θ_{old} to the new parameter θ_{new}. The update can be performed using the following expression:

$$\theta_{new} \leftarrow \theta_{old} + \eta\,\nabla_\theta J(\theta) \tag{3.5}$$

This update rule allows for the iterative refinement of the policy parameter θ based on the gradient of the objective function. By adjusting θ in the direction of the gradient, the policy search aims to find the optimal parameter that maximizes the expected cumulative rewards.

3.3 Generative Adversarial Nets

Text generation is a fascinating and challenging task in NLP. GANs have emerged as a powerful approach to tackling this problem. GANs, initially proposed in [18], have primarily been associated with image-generation tasks. However, their application to text generation has garnered significant attention and shown promising results.

Traditional approaches to text generation, such as LMs GANs based on RNNs, often rely on maximizing the likelihood of generating the next word given the previous context. While these models can generate coherent and grammatically correct sentences, they may struggle with generating diverse and creative text. GANs, on the other hand, provide an alternative paradigm that promotes diversity and realism in the generated text [28].

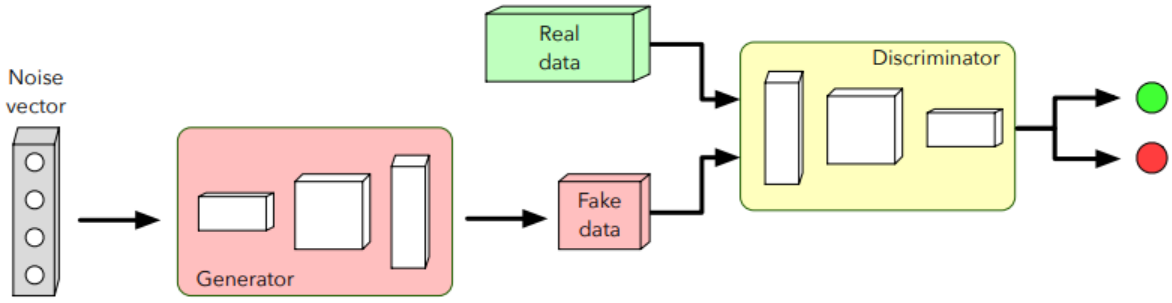

Figure 3.2: Standard architecture of a GAN [28]

In the context of text generation, GANs consist of two main components: a generator network and a discriminator network. The generator network takes random noise as input and generates synthetic text samples. The discriminator network, similar to a binary classifier, distinguishes between real text data from a training set and the synthetic text generated by the generator [28]. Figure 3.2 illustrates an example of a standard GAN.

The network's discriminator D is trained to maximize the probability of classifying both training and generated data as real images. Simultaneously, the network's generator G is trained to

minimize $\log(1 - \mathrm{D}(\mathrm{G}(\mathrm{z})))$, i.e., the divergence among both data distributions, where z denotes a noisy input [28]. In other words, the two neural-networks system compete in a zero-sum game, represented by Equation 3.6:

$$\min_{\mathrm{G}} \max_{\mathrm{D}} \mathrm{C}(\mathrm{D}, \mathrm{G}) = \mathbb{E}_x\left[\log(\mathrm{D}(\mathrm{x}))\right] + \mathbb{E}_z\left[\log(1 - \mathrm{D}(\mathrm{G}(z)))\right] \tag{3.6}$$

Where $\mathrm{C}(\mathrm{D}, \mathrm{G})$ is the loss function to be minimized, $\mathrm{D}(\mathrm{x})$ is the discriminator's estimated probability of a real sample x being real, $\mathbb{E}_x$ is the mathematical expectancy overall samples from the real data set $\mathcal{X}$, $\mathrm{G}(z)$ stands for the generated fake data given the noise vector z, $\mathrm{D}(\mathrm{G}(z))$ is the estimated probability of a fake sample $\mathrm{G}(z)$ being real, and $\mathbb{E}_z$ is the mathematical expectancy over all random generator inputs, i.e., the expected value over all fake samples generated by G [28].

However, a deriving problem from Equation 3.6 is that GANs can get trapped in local optimums when the discriminator has an easy task. In other words, at the beginning of the training, when G still does not generate proper samples, D can reject fake samples with high probability, as they are incredibly different from the real data. Thus, an alternative to such a problem is to train G to maximize $\log \mathrm{D}(\mathrm{G}(\mathrm{z}))$, which improves the initial gradient values [28].

The discriminator in GANs is a binary classifier represented as a neural network. Given an input sample x, the discriminator outputs a probability value between 0 and 1, representing the likelihood that x belongs to the real data distribution. The objective of the discriminator is to maximize the probability of assigning high values to real data while minimizing the probability of assigning high values to fake data generated by the generator.

During the training process, the generator and discriminator engage in a competitive game. The generator aims to produce text that can fool the discriminator into classifying it as real, while the discriminator aims to accurately distinguish between real and fake text samples. Through this adversarial interaction, the generator progressively improves its ability to generate more realistic and diverse text, while the discriminator becomes more adept at discerning between real and synthetic text [28].

Algorithm 1 Minibatch stochastic gradient descent training of GANs. The number of steps to apply to the discriminator, k is a hyperparameter [18].

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\mathrm{z}^{(1)}, ..., \mathrm{z}^{(\mathrm{m})}\}$ from noise prior $\mathrm{p_g}(\mathrm{z})$.
  - Sample minibatch of m examples $\{\mathrm{x}^{(1)}, ..., \mathrm{x}^{(\mathrm{m})}\}$ from data generating distribution $\mathrm{p_{data}}(\mathrm{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{\mathrm{m}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \left[\log \mathrm{D}(\mathrm{x}^{(\mathrm{i})}) + \log\left(1 - \mathrm{D}\left(\mathrm{G}(\mathrm{z}^{(\mathrm{i})})\right)\right)\right]$$

  end for

  - Sample minibatch of m noise samples $\{\mathrm{z}^{(1)}, ..., \mathrm{z}^{(\mathrm{m})}\}$ from noise prior $\mathrm{p_g}(\mathrm{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

The success of GANs in text generation relies on their ability to capture the underlying distribution of the training text data and generate samples that align with that distribution. GANs have demonstrated their capacity to generate coherent sentences, capture semantic relationships, and exhibit creativity in text-generation tasks. They have been employed for diverse applications such as dialogue generation, story generation, machine translation, and poetry generation.

3.3.1 Sequence Generative Adversarial Nets

SeqGAN is a variant of GANs specifically designed for sequence data, including text generation. SeqGAN introduces a RL framework to address the challenge of training GANs for sequential data, where the traditional GAN training process can be unstable [29].

In SeqGAN, the generator network is responsible for generating sequential data, such as sentences or text paragraphs, while the discriminator network evaluates the generated sequences and provides feedback. The key idea behind SeqGAN is to treat the generation process as a sequential decision-making problem and apply RL to train the generator [29].

The Generator (G) in SeqGAN is typically an RNN-based model, such as LSTM or GRU. It takes a random noise vector z as input and produces a sequence $x = (x_1, x_2, ..., x_T)$ where each x_T represents a token in the vocabulary. During training, the objective of the generator is to maximize the probability of generating sequences that look like real text data. However, directly optimizing the probability of generating high-quality sequences is challenging due to the discrete nature of text.

To deal with this challenge, SeqGAN introduces the Discriminator (D), which is also an RNN-based model like the generator. The discriminator's role is to assess the realism of the generated sequences. It takes a sequence x as input and outputs a probability $D(x)$, indicating how likely x is a real (i.e., human-written) sequence.

The training process of SeqGAN unfolds in several steps, incorporating both adversarial training and Monte Carlo search.

- **Pre-training:** First, the Generator undergoes pre-training using Maximum Likelihood Estimation (MLE) on the training data. This step helps the generator learn to produce sequences that are likely to occur in the training data.
- **Adversarial Training with Monte Carlo Search:** Following pre-training, the adversarial training with Monte Carlo search begins. Here is where the GAN dynamics, Policy Gradient, and Monte Carlo search come into play:

The Generator starts generating sequences x from noise vectors z, which are then evaluated by the Discriminator. The Discriminator provides a reward signal, referencing how realistic the generated sequence appears. This reward serves as the basis for training the generator further. The objective of the generator is to maximize the expected reward, supporting it to produce sequences that the discriminator finds difficult to distinguish from real data:

$$J(G) = \mathbb{E}_{x\sim G}[\log D(x)] \tag{3.7}$$

However, since the generator outputs sequences and not just a single value, traditional gradient methods cannot be directly applied. This is where the Monte Carlo search comes in. The generator samples K sequences $x^1, x^2, ..., x^K$ from the current policy G, and then calculates the expected reward over these sequences:

$$J(G) = \frac{1}{K}\sum_{k=1}^{K}[\log D(x^k)] \tag{3.8}$$

This Monte Carlo estimation helps in approximating the expected reward over multiple sequences, providing a more stable and accurate update for the generator's parameters. Concurrently, the Discriminator is trained to classify sequences as real or fake. Real sequences from the training data are labeled as $D(x) = 1$, while generated sequences from the generator are labeled as $D(x) = 0$. The discriminator aims to maximize the probability of correct classification:

$$J(D) = -\mathbb{E}_{x\sim \text{Real Data}}[\log D(x)] - \mathbb{E}_{x\sim G}[\log(1 - D(x))] \tag{3.9}$$

Throughout training, the generator and discriminator share in an adversarial process. The generator learns to produce more realistic sequences to get higher rewards from the discriminator, while the discriminator improves its ability to distinguish between real and fake sequences. This iterative process continues until both networks reach a balance, where the generator produces realistic sequences that fool the discriminator, and the discriminator cannot effectively differentiate between real and fake sequences.

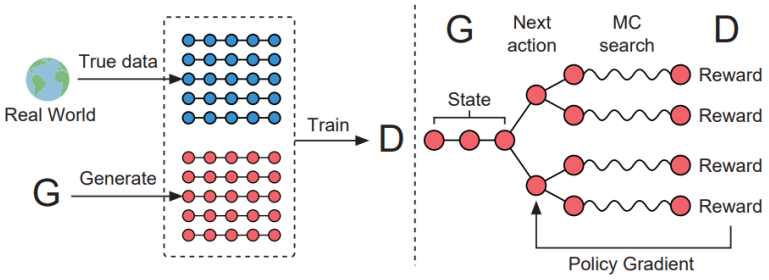

Figure 3.3: The illustration of SeqGAN [29]

Figure 3.3 provides an illustration of the training process in SeqGAN. On the left side, the discriminative model D is updated by providing positive examples from real data and negative examples from the sequences generated by the generative model G. This training process enables D to distinguish between real and generated sequences.

On the right side, the generative model G is trained using a policy gradient approach. The final reward signal is provided by the discriminative model D and is used to update the generator's parameters θ. This update is performed via the policy gradient method, which maximizes the expected reward received from D. Additionally, the intermediate action value is refined using Monte Carlo search, allowing the generator to explore the sequence space more effectively [29].

The training of SeqGAN involves a feedback loop between the discriminative and generative models. D guides the improvement of G by providing rewards based on the likelihood of fooling D. The generative model G, in turn, seeks to generate sequences that maximize the reward signal received from D. This iterative process leads to the refinement of both the generative and discriminative models, ultimately resulting in the generation of high-quality and realistic sequences [29].

The combination of policy gradient and MC search is a key aspect of the model's training process. Here is a detailed overview of how these components collaborate:

1. **Policy Gradient**: SeqGAN utilizes a policy gradient method, inspired by RL. In RL, the goal is to find a policy (a strategy) that maximizes the expected cumulative reward. In the context of SeqGAN, the "policy" refers to the generator's strategy for producing sequences of words.
2. **MC Search:** It search is employed to estimate the expected reward associated with a generated sequence. It involves sampling several sequences from the generator and using them to estimate the average reward.
3. **Combining Policy Gradient and MC Search:** The policy gradient is used to update the parameters of the generator's neural network. This involves adjusting the weights to increase the probability of generating sequences that result in higher rewards. MC search is employed to approximate the expected reward. Multiple sequences are generated using the current generator policy, and these sequences are then evaluated by a discriminator (critic) to estimate their quality.
4. **Training Procedure:** The training procedure involves two main steps:
   a. **Roll-out:** several complete sequences are sampled from the current generator policy. These sequences are generated without updating the generator's parameters.
   b. **Update:** The generator's parameters are updated using the policy gradient. The update is based on the sampled sequences and their estimated rewards obtained through MC search.
5. **Objective Function:** The objective is to maximize the expected reward for the generated sequences. The policy gradient is applied to adjust the generator's parameters in the direction that increases the expected reward.
6. **Iterative Improvement:** The process is iteratively repeated, with the generator gradually improving its policy by receiving feedback from the discriminator and adjusting its parameters accordingly.

In summary, policy gradient guides the generator to improve its strategy for generating sequences, while Monte Carlo search is employed to estimate the expected rewards associated with those sequences. This combination allows SeqGAN to iteratively enhance its performance in generating high-quality sequences.

3.3.2 Text Generative Adversarial Nets

The authors of the paper [20] introduced TextGAN as a model designed for generating realistic sentences using an adversarial training scheme. The model utilizes an LSTM architecture as the generator and a CNN as the discriminator. The adversarial training involves discriminating generated text from real text, resulting in a holistic training perspective. This approach ensures that the generated sentences maintain high quality throughout the entire generation process.

The Generator (G) in TextGAN is often an LSTM-based model. It takes a random noise vector z as input and generates a sequence of tokens representing text. This initial sequence is then passed through the CNN layer for feature extraction. The CNN's role is definitive for capturing hierarchical patterns in text data, especially in tasks like sentence generation where local and global contexts are important.

The CNN runs on the token sequence in a convolutional manner, similar to how CNNs process images but suitable for text. It consists of convolutional layers that slide a filter over the token sequence to extract local features. After convolution, pooling layers such as max-pooling are often used to reduce dimensionality and capture the most important features. The output from the pooling layers is then flattened to be fed into a fully connected layer.

The Discriminator (D) in TextGAN is typically an LSTM-based model combined with the CNN's output. The Discriminator evaluates the features extracted by the CNN to determine if the text is real or generated. It aims to classify sequences of tokens as either real (from the training data) or fake (generated by the Generator).

The training process of TextGAN involves an adversarial training procedure, similar to traditional GANs. The Generator and Discriminator are trained iteratively, each aiming to outperform the other. The Generator initially takes random noise z and generates a sequence of tokens. This sequence is then passed through the CNN layers for feature extraction. The features from the CNN are used as input to the Discriminator for evaluation. The Generator's objective is to produce text that fools the Discriminator into classifying it as real.

The Discriminator, on the other hand, is trained to correctly classify text sequences as real or fake. It receives real text sequences from the training data and fake sequences from the Generator. The Discriminator learns to assign high probabilities to real sequences and low probabilities to fake ones. The adversarial training process continues iteratively, with both the Generator and Discriminator improving their performance. The Generator is trained to maximize the probability that the Discriminator assigns to the generated text being real.

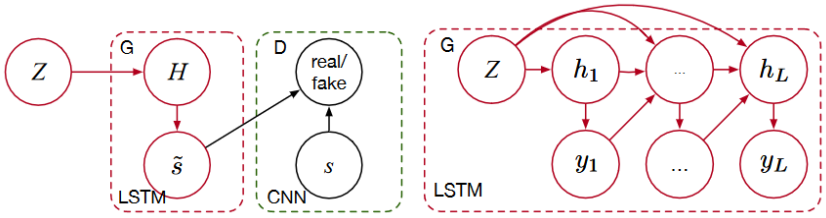

Figure 3.4: Model scheme of TextGAN [20]

In Figure 3.4, the TextGAN model takes a corpus $S = \{s_1, \cdots, s_n\}$, where n is the total number of sentences. Let w^t denote the t-th word in sentences s. Each word w^t is embedded into a k-dimensional word vector $x_t = \mathcal{W}_e[w^t]$, where $\mathcal{W}_e \in \mathbb{R}^{k\times V}$ is a word embedding matrix (to be learned), V is the vocabulary size, and notation $[v]$ denotes the index for the v-th column of a matrix.

The CNN discriminator in the TextGAN model is employed for sentence encoding. This involves a convolutional layer and a max-pooling operation applied to the entire sentence for each feature map. To represent a sentence of length T, a matrix $X \in \mathbb{R}^{k\times T}$ is formed by concatenating its word embeddings as columns, where the t-th column is denoted as x_t. The convolution operation utilizes a filter $\mathcal{W}_c \in \mathbb{R}^{k+h}$, applied to a window of h words, generating a new feature. This process is expressed as $c = f(X * W_c + b) \in \mathbb{R}^{T-h+1}$, where $f(\cdot)$ is a nonlinear activation function like the hyperbolic tangent, $b \in \mathbb{R}^{T-h+1}$ is a bias vector, and * denotes the convolutional operator.

Convolving the same filter with the h-gram at every position in the sentence allows features to be extracted independently of their position. Subsequently, a max-over-time pooling operation is applied to the feature map, and the maximum value, $\hat{c} = \max\{c\}$, is taken as the feature corresponding to that particular filter. This process helps capture the most salient features in the sentence, contributing to the discrimination process performed by the CNN discriminator in TextGAN.

The pooling scheme employed in TextGAN's CNN discriminator aims to capture the most crucial feature, signified by the highest value, for each feature map. This approach effectively filters out less informative word compositions, ensuring that the extracted features are representative of the most significant elements in the sentence. Additionally, this pooling scheme guarantees that the extracted features remain independent of the input sentence's length.

In practical terms, TextGAN uses multiple filters with varying window sizes to extract a multitude of features. With m window sizes and d filters for each size, the model obtains an md-dimensional vector (vector feature layer) denoted as f, which effectively represents a sentence. The use of diverse filters enhances the model's ability to understand patterns and features across different scales.

The model uses a softmax layer to categorize the input sentence. This layer maps the input to an output D(X) in the range of 0 to 1. This output references the probability of the input sentence belonging to the real data distribution instead of being generated by an adversarial generator.

For sentence generation, TextGAN employs an LSTM generator (decoder part) that translates a latent vector z into the synthetic sentence s̃. The probability of generating a length-T sentence s̃ given the encoded feature vector z is defined using the conditional probability formula:

$$p(\tilde{s} \mid z) = p(w^1 \mid z) \prod_{t=2}^{T} p(w^t \mid w^{<t}, z) \tag{3.11}$$

This formula captures the probability of each word in the generated sentence conditioned on the previous words and the latent vector z, contributing to the overall coherence and contextuality of the synthetic sentence.

They generate the first word w^1 from z, with $p(w^1 \mid z) = \text{argmax}(Vh_1)$, this calculates the probability distribution over the vocabulary for the first word w^1. V is a weight matrix used for computing a distribution over words, the output of the softmax is used to sample the first word. $h_1 = \tanh(Cz)$, this is the computation for the hidden state at the first time step, where C is a weight matrix, and z is the noise vector. Bias terms are omitted for simplicity. All other words in the sentence are then sequentially generated using the RNN until the end-sentence symbol is generated. Each conditional $p(w^t \mid w^{<t}, z)$, where $< t = \{1, \ldots, t-1\}$, is specified as $\text{argmax}(Vh_t)$, where h_t, the hidden units, are recursively updated through $h_t = \mathcal{H}(y_{t-1}, h_{t-1}, z)$. The LSTM input y_{t-1} for the t-th step is the embedding vector of the previous word that maximizes the w^{t-1}.

$$y_{t-1} = W_e[w^t] \tag{3.12}$$

In summary, the iterative nature of adversarial training in TextGAN involves a continuous cycle of improvement, with both the generator and discriminator refining their abilities through mutual interaction. This iterative refinement is key to achieving the generation of high-quality and realistic text.

3.3.3 Rank Generative Adversarial Nets

The generative model proposed in [23] is called RankGAN. In RankGAN, the Generator (G) is typically an LSTM-based model that takes random noise vectors z as input and generates sequences of tokens representing text. These sequences, initially unranked, are the output of the Generator and are based on the randomness of the noise vectors. The objective of the Generator is to produce realistic and high-quality text sequences that are indistinguishable from a reference set of real data, often referred to as the "Reference".

A key component of RankGAN is the Ranker (R), which provides pairwise comparisons and rankings of the generated sequences. The Ranker evaluates the quality of these sequences by comparing them to each other, aiming to learn a ranking function that accurately ranks sequences based on their quality. During training, the Ranker receives pairs of sequences generated by the Generator and computes the probability that it correctly ranks one sequence over the other. The

Ranker's loss function is designed to support accurate pairwise rankings, guiding it to provide rankings that align with the Discriminator's scores.

The Ranked Sentences refer to the output of the Ranker after it has evaluated the generated sequences. Based on the Ranker's evaluations, the sequences are ranked from highest to lowest quality. These rankings provide valuable feedback to the Generator, guiding it to improve the quality of its generated text. The training process in RankGAN is adversarial, similar to traditional GANs. The Generator aims to produce text that not only fools the Discriminator (D) but also ranks high according to the Ranker. This adversarial training loop continues iteratively, with both the Generator and Ranker improving their performance over time.

Figure 3.5 shows the inclusion of the ranker introduces a different dynamic to the adversarial training process, allowing for an assessment of the quality and authenticity of generated sequences.

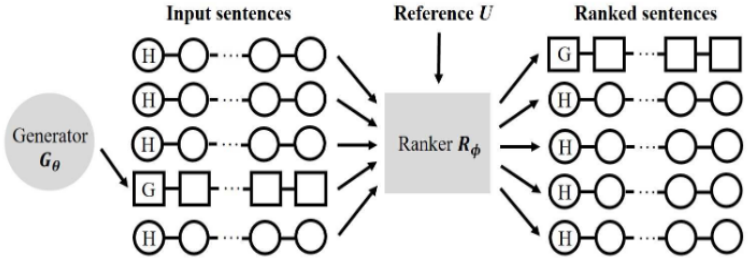

Figure 3.5: Model scheme of RankGAN [23]

In the RankGAN model, the learning objective for the generator is to produce synthetic sentences that receive higher ranking scores than those drawn from real data. On the other hand, the ranker aims to rank the synthetic sentences lower than human-written sentences. These adversarial dynamic results in G and R engaging in a minimax game with the following objective function L:

$$\min_{\theta}\max_{\phi}\mathcal{L}(G_\theta,R_\phi) = \mathbb{E}_{s\sim P_h}\left[\log R_\phi(s|U,C^-)\right] + \mathbb{E}_{s\sim G_\theta}\left[\log\left(1-R_\phi(s|U,C^+)\right)\right] \quad (3.14)$$

Where θ and ϕ are the variable parameters in G and R, respectively. The $\mathbb{E}$ is the expectation operator, and P_h is the real data from human-written sentences. $s\sim P_h$ and $s \sim G_\theta$ denote that s is from human-written sentences and synthesized sentences, respectively. The U is the reference set used for estimating relative ranks, $C^+$and C^- are the comparison set about different input sentences s. When the input sentence s is the real data, C− contains generated data pre-sampled from G_θ. If the input sentence s is the synthetic data, the human-written data is pre-sampled and enclosed in C+. The forms of G_θ and R_ϕ can be achieved in many ways.

The LSTM model iteratively processes the embedded features of the current tokenw_t, along with information from the previous hidden state h_{t-1} and cell statc_{t-1}, to update the current states h_t and c_{t-1}. The subsequent word w_{t+1} is conditionally sampled based on the probability

distribution $p(w_{t+1}| h_t)$, determined by the current hidden state h_t. Leveraging the capabilities of LSTM, the generative model retains long-term gradient information, enabling the production of more intricate word sequences $s = (w_0, w_1, w_2, ..., w_t)$.

The proposed ranker R, sharing a similar convolutional architecture with the discriminator in TextGAN, initially maps concatenated sequence matrices into embedded feature vectors $y_s = F(s)$ through a series of nonlinear functions F. Subsequently, the ranking score is calculated for the sequence features y_s concerning the reference feature y_u, extracted by R in advance.

The ranking score for an input sentence is an expectation of its scores given different references sampled across the reference space. During learning, the randomly sample a set of references from human-written sentences to construct the reference set U. Meanwhile, the comparison set C will be constructed according to the type of the input sentence s, C is sampled from the human-written set and machine-generated set. With the above setting, the expected ranking score computed for the input sentence s can be derived by:

$$R_\phi(s \mid U,C) = \mathbb{E}_{u \in U}[P(s \mid u,C)] \tag{3.15}$$

Here, s is the input sentence. It is either human-written or produced by G_θ. Accordingly, u is a reference sentence sampled from set U. Given the reference set and the comparison set to compute the rank scores indicating the relative ranks for the complete sentences. The ranking scores will be used for the objective functions of generator G_θ and rankerR_ϕ.

Chapter Four

Methodology

In this chapter, we delve into the methodology employed in our study, outlining the stages of implementation and detailing the research design. Commencing with an overview of the computing environment utilized in our experiments, we proceed to introduce the TextBox tools, a pivotal framework facilitating text generation tasks. TextBox Tools, a versatile toolbox, played a crucial role in running the GANs models for unconditional text generation, allowing easy experimentation with various parameters and data configurations. This flexibility is particularly valuable for optimizing models for Arabic language generation.

We provide an in-depth exploration of the architectures of SeqGAN, TextGAN, and RankGAN models within TextBox tools, offering comprehensive insights into the hyperparameters influencing model behavior. The parameters optimized for SeqGAN are subsequently applied to other GAN models. Following this, we present the Arabic dataset employed in our study. Finally, we discuss the application of the Bilingual Evaluation Understudy (BLEU) metrics, outlining their role in evaluating the quality of generated Arabic sentences.

4.1 Computing Environment

The machine learning algorithms employed in our experiments were executed on an Ubuntu server running the CONDA 23.3.1 version (Miniconda), utilizing Python 3.10.9. This setup was established on Ubuntu 20.04.6 LTS, operating on an Intel® Xeon(R) W-2145 CPU clocked at 3.70GHz × 16, complemented by a 32 GB RAM memory. For our effort, we harnessed the capabilities of the TextBox-0.2.1 tools facilitated by the PyCharm IDE.

4.2 The Procedures

Over an extensive five-month period, we undertook more than 45 experiments, meticulously investigating a multitude of parameters for each model. Our initial step involved acquiring a substantial collection of articles from the Al-Arabiya website, focusing on the realms of culture and art. These articles underwent thorough processing, with segmentation into paragraphs and subsequent transformation into concise, accurate, and meaningful sentences. We remove symbols and English words to get short Arabic sentences.

With the prepared data in hand, we partitioned it into three files—train, valid, and test—each containing grammatically Arabic sentences. Transitioning to Textbox tools, we initiated the process of overseeing and checking parameters. The examination commenced by testing dataset parameters via a YAML file that is a human-readable data serialization format that stands out for its simplicity and readability. It is commonly used for configuration files, data exchange, and storing structured data in a way that is easy for both humans to read and write, as well as for machines to parse. Subsequently, our focus shifted to GAN parameters, we fine-tuning them through YAML files to arrive at optimal values for GAN models.

The test extended to SeqGAN parameters, TextGAN, and RankGAN, with each parameter systematically examined. Our testing methodology adhered to a step-by-step approach, testing each parameter individually for the dataset, GAN, SeqGAN, TextGAN, and RankGAN. Optimal values were methodically selected, fixed, and the process repeated until all parameters were thoroughly examined and optimized.

4.3 TextBox Tools and Framework

Text generation, an essential branch of NLP commonly known as natural language generation (NLG), it aims to produce plausible and understandable text in human language from input data like sequences or keywords. With the remarkable performance of deep learning models, the TextBox library was developed by [1] as an open-source solution to facilitate text generation tasks in NLP.

TextBox library aims to enhance reproducibility, standardize implementation and evaluation, and facilitate the development of new algorithms. The library offers a unified and modularized framework, built upon PyTorch, for easy comparison and customization of text generation models. These modules include the data module, model module, evaluation module, and various common components and functionalities [1].

TextBox offers a wide range of text generation models, including Variational Auto-Encoder (VAE), GAN, RNN, Transformer, and Pretrained Language Models (PLM). It supports various tasks like text summarization and machine translation, with flexible evaluation options using popular metrics like BLEU and ROUGE. In addition, TextBox provides convenient interfaces for various common functions and modules in text generation models, making it easy for users to build and evaluate their own models. The library's interfaces are fully compatible with PyTorch, allowing seamless integration of user-customized modules and functions [1].

Figure 4.1 presents the illustration of the main functionalities and modules in TextBox library. The configuration module at the bottom helps users set up the experimental environment like hyperparameters and running details. Built upon the configuration module, the data, model, and evaluation modules form the core elements of TextBox library.

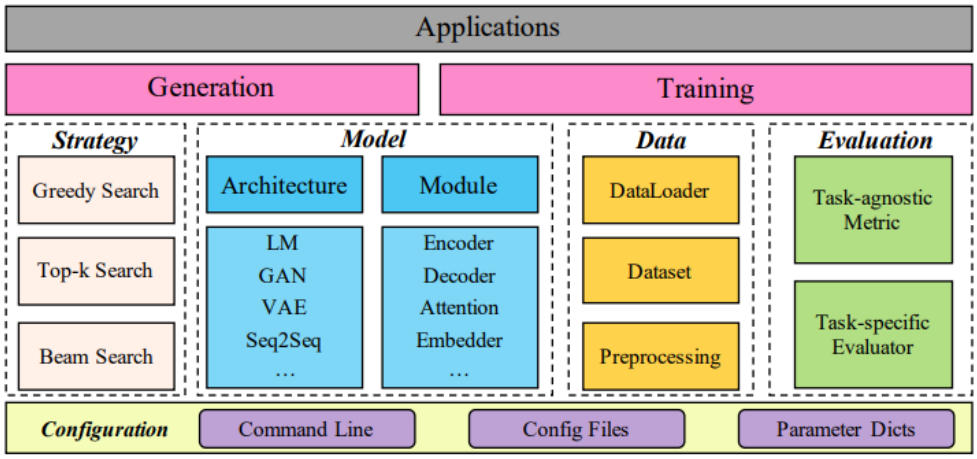

Figure 4.1: The illustration of the main functionalities and modules in TextBox library [1]

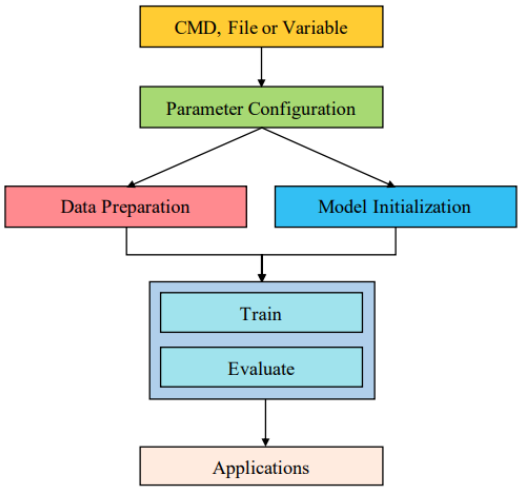

Figure 4.2: An illustractive usage flow of TxtBox library [1]

Figure 4.2 illustrates the general usage flow when running a model in TxtBox library. The procedure involves obtaining experimental configuration from files, the command line, or parameter dictionaries. The dataset and model are prepared and initialized based on the configured settings, and the execution module handles model training and evaluation. To utilize the unified Data and Evaluation modules, users need to implement a specific Model class and mandatory functions.

Implementing GANs for text generation involves several key steps. Here is a high-level overview of the process:

1. **Dataset Preparation:** Prepare a dataset of text samples that will serve as the training data for the GAN. This dataset should contain a large number of text sequences that the model can learn from.

2. **Generator Network:** Design the generator network, which takes random noise as input and generates synthetic text samples. The generator can be implemented using RNNs such as LSTM or GRU, or transformer-based architectures like GPT.
3. **Discriminator Network:** Design the discriminator network, which takes text samples as input and predicts whether they are real or fake. The discriminator can be implemented using RNNs, CNNs, or other architectures suitable for text classification.
4. **Training Loop:** Train the GAN in an alternating manner. In each training iteration:
    - Generate a batch of synthetic text samples using the generator.
    - Randomly sample a batch of real text samples from the training dataset.
    - Train the discriminator using both the real and synthetic samples, optimizing it to correctly classify real and fake text.
    - Freeze the discriminator weights and train the generator using the discriminator's output as feedback, aiming to generate text samples that fool the discriminator into classifying them as real.
5. **Loss Functions:** Define appropriate loss functions for the generator and discriminator. The generator's loss is typically based on the discriminator's output, aiming to maximize the probability of fooling the discriminator. The discriminator's loss involves distinguishing between real and fake text samples.
6. **Optimization:** Use an optimizer like Adam or RMSprop to update the generator and discriminator parameters based on their respective loss functions. Iterate this optimization process for a set number of epochs or until convergence.
7. **Evaluation:** Evaluate the quality of the generated text samples by assessing metrics such as text coherence, diversity, and relevance to the training data. You can also employ human evaluation or conduct automated evaluations using language models or classifiers.
8. **Fine-tuning and Iteration:** Fine-tune the GAN as needed by adjusting hyperparameters, modifying network architectures, or employing techniques like regularization, dropout, or attention mechanisms. Iterate this process to improve the quality of the generated text.

However, challenges remain in GAN-based text generation. Ensuring the generated text's grammatical correctness, coherence, and maintaining control over generated content are ongoing research areas. Techniques such as RL, attention mechanisms, and RL from human feedback have been explored to address these challenges and improve the quality and controllability of generated text.

Overall, GANs have emerged as a powerful approach to text generation, offering the potential to generate diverse, realistic, and creative scripts. In this research, we use GAN models and their applications in creating text in the Arabic language, to enhance and open new possibilities for generating Arabic texts that we use in various fields.

4.3.1 Unconditional Generation Task in TexBox Tools

In TextBox-0.2.1 tools, we implemented all experiments with unconditional task, this concept pertains to text generation tasks where the model is expected to generate text without any specific conditioning input. This means that the generated text is not reliant on any explicit context or prompt, but is produced in a free-form manner based on the learned patterns and knowledge from the training data.

Unconditional text generation tasks in TextBox-0.2.1 tools involve training models that can generate coherent and contextually relevant text sequences without being guided by any predefined input. This is in contrast to conditional text generation tasks, where the model generates text based on a given input or prompt.

The TextBox-0.2.1 library provides the necessary modules, components, and models to facilitate both unconditional and conditional text generation tasks. Researchers and practitioners can utilize these tools to develop and train models that excel in generating high-quality text in various contexts, including unconditional text generation scenarios.

4.4 SeqGAN Model Architecture in the TextBox Tools

The seqGAN model architecture in the TextBox-tools-0.1.2 library is designed to generate realistic and coherent text sequences through a combination of generative modeling, RL, and adversarial training. The technical aspects of the SeqGAN model architecture in TextBox tools:

- **Generator Network:**
  1. Embedding Layer: The embedding layer maps discrete word indices to continuous vectors. It is typically initialized with pre-trained word embeddings like Word2Vec or GloVe.
  2. Recurrent Layers: TextBox Tools supports various recurrent cell types, such as LSTM or GRU. These cells capture the sequential dependencies in the data. Hidden states are updated at each time step, encoding context information.
  3. Time-Step Outputs: At each time step, the output of the recurrent layer is a vector representing the predicted probabilities of the next word. This output is passed through a softmax function to obtain a valid probability distribution over the vocabulary.
  4. Sampling Strategy: Different sampling strategies can be employed during training and inference. "Greedy" sampling selects the word with the highest probability, while "top-k" and "nucleus" sampling introduce controlled randomness for more diverse outputs.

- **Discriminator Network:**
  1. Convolutional/Recurrent Layers: The discriminator network processes sequences using convolutional or recurrent layers. Convolutional layers apply multiple filters to capture local patterns, while recurrent layers maintain context and dependencies.

2. Feature Extraction: In convolutional layers, filters slide over the sequence, extracting features at different positions. In recurrent layers, hidden states capture contextual information over time.
3. Classification Layer: The final layer is typically a dense layer with a sigmoid activation, producing a probability score indicating whether the input sequence is real or generated.

- **Adversarial Training:**
  1. Generator Update: The generator aims to maximize the probability that the discriminator misclassifies its generated sequences. The generator's loss is the negative log-likelihood of the discriminator's prediction for generated sequences.
  2. Discriminator Update: The discriminator aims to correctly classify real and generated sequences. Its loss is computed using binary cross-entropy based on the actual labels (real or fake).
- **Loss Functions:**
  1. Generator Loss: The generator's loss L_G is calculated as the negative log-likelihood of the discriminator's prediction for generated sequences: $L_G = -\log(D(G(z)))$. Here, z is the noise input.
  2. Discriminator Loss: The discriminator's loss L_D combines the losses from real and generated sequences: $L_D = -\log(D(x)) - \log(1 - D(G(z)))$. Here, x is a real sequnce.
- **Hyperparameters and Optimization:**
  1. TextBox Tools allows you to set hyperparameters such as learning rates, batch sizes, optimizer settings, and more.
  2. Popular optimizers like Adam can be chosen, and learning rates can be scheduled or decayed during training.
- **Sampling Strategies and Temperature:**
  1. Temperature: The temperature parameter adjusts the output distribution from the generator. Higher values make the distribution more uniform, resulting in more randomness.
  2. Sampling Strategies: Techniques like top-k sampling select from the top-k most probable words at each step, enhancing diversity. Nucleus sampling considers the top-p fraction of words.
- **TensorBoard Integration:** TextBox Tools often integrates with TensorBoard to visualize various aspects of training. You can monitor generator and discriminator losses, generated samples, and other relevant metrics.
- **Evaluation Metrics:** SeqGAN models in TextBox Tools can be evaluated using metrics like BLEU, ROUGE, perplexity, and self-BLEU. These metrics help quantify the quality and diversity of generated text.

In summary, TextBox Tools' SeqGAN model architecture is a combination of a sophisticated generator network, a discriminator network, and an adversarial training process. It involves intricate hyperparameter tuning, loss functions, and sampling strategies to achieve the dual goal of generating high-quality and diverse text while training the generator to improve over time.

SeqGAN has specific hyperparameters that influence its behavior and training dynamics, you can see these in an appendix.

4.5 TextGAN Model Architecture in the TextBox Tools

The TextGAN model architecture in the TextBox-tools-0.1.2 library is designed to generate realistic and coherent text sequences through a combination of generative modeling, RL, and adversarial training. The technical aspects of the TextGAN model architecture in TextBox tools:

- **Generative Modeling:** The model employs a generative approach to create new text sequences. This involves learning the underlying patterns and structures of the training data, enabling the generation of novel and contextually relevant text.

- **RL:** RL mechanisms are integrated to enhance the model's ability to generate high-quality sequences. This involves training the model through a reward-based system, where it receives positive reinforcement for generating desirable text.

- **Adversarial Training:** It is a crucial component, involving a dual learning process with a generator and a discriminator. The generator aims to create text that is indistinguishable from real data, while the discriminator learns to differentiate between genuine and generated text. This adversarial setting refines the model's text generation capabilities.

- **Hyperparameter Configuration:** The model architecture includes a set of hyperparameters that dictate its behavior during training and generation. These parameters, such as generator_embedding_size, discriminator_embedding_size, hidden_size, dropout_rate, l2_reg_lambda, mmd_lambda, recon_lambda, filter_sizes, filter_nums, and adversarial_g_epochs, are configured to optimize performance.

- **Sequence Generation Process:** The sequence generation process involves the systematic unfolding of the model over time. The generator produces text sequences step by step, considering the context and learned patterns to ensure coherence and relevance in the generated content.

- **Embedding Spaces:** Dimensionality considerations are crucial in the embedding spaces for both the generator and discriminator. These spaces define the size of word vectors and contribute to the richness of representation in the generated and real data.

- **Regularization Techniques:** To prevent overfitting and enhance generalization, the model incorporates regularization techniques such as dropout_rate and l2_reg_lambda. These mechanisms contribute to the stability and robustness of the training process.

- **Diversity Enhancement:** Maximum Mean Discrepancy (MMD) is integrated into the loss function, providing a mechanism to enhance diversity in the generated samples. This ensures that the model produces a varied and contextually diverse set of text sequences.

The TextGAN model architecture, embedded within the TextBox-tools-0.1.2 library, intricately combines generative modeling, RL, and adversarial training to facilitate the creation of realistic and coherent text sequences. Leveraging generative approaches, the model learns underlying patterns from training data, employing RL for reward-based training and adversarial training for refining text generation.

Configurable hyperparameters, including dimensions for embedding spaces and regularization techniques, play a pivotal role in optimizing performance. The sequence generation process unfolds systematically over time, considering context and learned patterns. The incorporation of diversity-enhancing mechanisms, such as Maximum Mean Discrepancy (MMD), ensures the generation of varied and contextually diverse text sequences. This nuanced architecture represents a sophisticated framework for robust text generation in the Arabic language, necessitating an understanding of its technical intricacies for effective utilization.

4.6 RankGAN Model Architecture in the TextBox Tools

The RankGAN model architecture in the TextBox-tools-0.1.2 library is crafted to produce realistic and coherent text sequences, employing a fusion of generative modeling, RL, and adversarial training. The technical intricacies of the RankGAN model architecture in TextBox tools include:

The RankGAN model architecture, integrated into the TextBox-tools-0.1.2 library, is meticulously crafted to produce realistic and coherent text sequences through the amalgamation of generative modeling, RL, and adversarial training. The technical intricacies of the RankGAN model within TextBox tools unfold in several key aspects:

- **Generator and Discriminator Configuration:** The model incorporates a generator responsible for synthesizing text and a discriminator trained to distinguish between real and generated text. This adversarial setup is fundamental to refining the generative process.
- **Embedding Size and Hidden Dimensions:** Configurable parameters such as the embedding size and dimensions of hidden states significantly impact the expressive capacity of the model. These dimensions influence how the model represents and processes information during text generation.
- **Dropout Mechanisms and Regularization:** The model employs dropout mechanisms during training, randomly dropping out neurons, and regularization techniques to mitigate overfitting. These measures enhance the generalization ability of the model to diverse linguistic patterns.
- **Monte Carlo Sampling Techniques:** The inclusion of Monte Carlo sampling contributes to the diversity of generated text. By employing multiple samples and selecting higher-ranked ones, the model achieves a more nuanced and varied output.

- **Ranking Loss and Adversarial Training:** Ranking loss is integrated into the training process, guiding the generator to produce text samples that rank higher. Adversarial training supplements this by refining generated text through a discriminative approach, making the model adept at capturing nuanced patterns.

- **Parameter Tuning Flexibility:** The model offers flexibility through tunable parameters, including learning rates, filter sizes, and the number of filters in convolutional layers. This adaptability allows practitioners to fine-tune the model for specific text generation tasks.

The RankGAN model architecture, embedded in the TextBox-tools-0.1.2 library, it designed for the generation of realistic and coherent text sequences. By combining generative modeling, RL, and adversarial training, the model achieves a sophisticated understanding of language patterns. Key technical include the configuration of the generator and discriminator, the influence of embedding size and hidden dimensions on expressiveness, the use of dropout mechanisms and regularization for overfitting mitigation, and the incorporation of Monte Carlo sampling for text diversity.

Additionally, the model leverages ranking loss and adversarial training to refine generated text, enhancing its ability to capture nuanced patterns. With flexible parameter tuning options, including learning rates and convolutional layer specifications, the RankGAN model stands as a versatile tool for tailoring text generation to specific linguistic tasks. This insight into the model's inner workings illuminates its capacity to generate diverse, contextually relevant text, positioning it as a valuable asset for NLP applications.

4.7 Dataset

We used an Arabic dataset containing cultural articles, sized between 400 MB and 500 MB, to train models in TextBox tools. Before starting training, we prepared the dataset using the transformers library. This step involved removing symbols, links, white spaces, and non-Arabic characters, so the dataset only had accurate Arabic text. The tqdm library was also used to split paragraphs into correct Arabic sentences, each placed on its own line in the main.tgt file.

This process resulted in a main.tgt file containing many accurate Arabic sentences, one on each line. Then we split the main.tgt file into three files (train.tgt, test.tgt, and vaild.tgt) are integral components of the machine learning process, particularly in the context of text generation tasks. These files play distinct roles in training and evaluating models and are crucial for achieving robust and effective text generation results.

In summary, training, validation, and testing files in TextBox tools serve distinct purposes in the Machine Learning pipeline. The training file is used to teach the model patterns and structures from a diverse set of data. The validation file helps in monitoring the model's performance during training to prevent overfitting. The testing file assesses the model's ability to generalize to new and unseen data. These files collectively ensure that the developed text generation model is robust, reliable, and capable of generating high-quality text in a variety of contexts.

4.8 Text Evaluation Using Bilingual Evaluation Under-Study (BLEU)

Bilingual Evaluation Understudy (BLEU) is indeed a metric used to evaluate the quality of generated sentences in comparison to reference sentences. Originally designed for machine translation systems, BLEU calculates the similarity between the machine-translated text and the reference text based on N-gram matching. Unigrams (single words) are treated as individual words, while bigrams (word pairs) are considered as single units [30].

The basic idea behind BLEU is to count the number of matching N-grams (unigrams, bigrams, trigrams, etc.) between the generated text and the reference data. Essentially, it is calculated by the co-occurrence of n-grams between the reference and candidate sentences, computing a type of precision between the sentences [31]. Let $\Phi = \{\Phi_1,\Phi_2,...,\Phi_k\}$ be the reference sentences with k tokens, while $\tilde{\Phi} = \{\tilde{\Phi}_1,\tilde{\Phi}_2,...,\tilde{\Phi}_w\}$ is the candidate sequence with w tokens. Equation 4.1 it is formula, as follows:

$$p_n = \frac{\sum_k \min\left(d_k(\tilde{\Phi}), \max(d_k(\Phi))\right)}{\sum_k d_k(\tilde{\Phi})} \tag{4.1}$$

Where $d_k(\cdot)$ is the amount of occurrences of an n-gram and k is the maximum number of possible n-grams. However, such formula inflates the metric's value for short sentences and needs a penalization factor according to Equation 4.2:

$$BP = \begin{cases} 1 & \text{if } w > k \\ e^{(1-k/w)} & \text{if } w \leq k \end{cases} \tag{4.2}$$

Therefore, the final BLEU score is calculated by a geometric mean weighted by the individual's n-grams precision. Moreover, the metric's value is within the [0, 1] interval, where values close to 1 indicate a bigger similarity between the sentences. Equation 4.3 describes such procedure, as follows:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{4.3}$$

Where n = 1, 2, 3. . . N and w_n is a constant for all n values. We utilized the fastBleu library to obtain the BLEU metric values.

The fastBleu library is a specialized software tool designed to swiftly compute the BLEU score, which is a widely used metric for evaluating the quality of machine-generated text against reference or human-written text. Think of fastBleu as a specialized tool in a programmer's toolkit, built to handle the specific task of calculating BLEU scores in a highly efficient manner.

When it comes to evaluating the performance of NLP models, such as text generators, BLEU scores provide valuable insights. They measure how well the generated text aligns with human

references and serve as a proxy for gauging the quality and fluency of the generated content. This is crucial for fine-tuning and optimizing the model.

FastBleu is like a calculator tailored for BLEU scores. Just as a regular calculator lets you perform mathematical operations with ease, fastBleu simplifies the complex calculations involved in assessing the quality of the generated text. It compares the generated text with the reference text, taking into account the precision of n-grams (word sequences) and their corresponding weights. This calculation process can be time-consuming and intricate, especially when dealing with large amounts of text data. However, fastBleu streamlines this process, making it quicker and more manageable.

By using fastBleu, programmers and researchers can easily integrate BLEU score calculations into their projects without needing to delve into the intricacies of the underlying mathematics. This saves time and effort, allowing them to focus on improving their text generation models based on the feedback provided by BLEU scores. In summary, fastBleu is an essential tool that empowers programmers to evaluate and enhance the quality of machine-generated text by efficiently calculating BLEU scores. Certainly, let us break down the process of calculating BLEU scores for the generated Arabic sentences:

1. **Tokenization:** The train and test files contain Arabic sentences. The first step is to tokenize these sentences into n-grams, where 'n' represents the desired level of granularity (1 for unigrams, 2 for bigrams, and so on).
2. **Precision Calculation:** For each sentence in the test file, the precision of n-grams is calculated concerning the corresponding sentences in the train file. This involves counting the number of overlapping n-grams in the generated sentence (test) compared to the reference sentence (train).
    - **BLEU-1:** It considers unigrams (single words) and calculates the precision of unigrams in the generated text compared to the reference text.
    - **BLEU-2:** It considers bigrams (pairs of adjacent words) and calculates the precision of bigrams in the generated text compared to the reference text.
    - **BLEU-3:** It considers trigrams (sequences of three adjacent words) and calculates the precision of trigrams in the generated text compared to the reference text.
    - **BLEU-4:** Similar to the others, but for four-grams.
3. **Modified Precision:** Precision is calculated as the number of overlapping n-grams in the generated text divided by the total number of n-grams in the generated text. A modification is introduced to the precision calculation to handle cases where the length of the generated text is significantly shorter than the reference. This is known as the "Brevity Penalty."
4. **Brevity Penalty (BP):** The brevity penalty is calculated based on the length of the generated sentence w and the effective reference length k. This ensures that the precision scores are appropriately adjusted to account for differences in length, see Equation 4.2.

5. **BLEU Score for Each Sentence:** The BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores are obtained for each sentence in the test file, see Equation 4.3.
6. **Average BLEU Score:** The final evaluation involves taking the average of the BLEU scores across all sentences in the test file. This provides an overall assessment of the quality of machine-generated Arabic sentences relative to the reference sentences in the train file.

Let us explain an example of how BLEU works to evaluate the quality of a machine-generated Arabic sentence compared to a human-generated reference.

- Candidate (Machine-Generated) Arabic Sentence: "القطة تجلس على السجادة."
- Reference (Human-Generated) Arabic Sentence: "القطة تجلس على السجادة في الصالة."

Breakdown into N-grams:

1. **Unigrams (1-gram):**
   - Candidate: "القطة", "تجلس", "على", "السجادة"
   - Reference: "القطة", "تجلس", "على", "السجادة", "في", "الصالة"
2. **Bigrams (2-gram):**
   - Candidate: "القطة تجلس", "تجلس على", "على السجادة"
   - Reference: "القطة تجلس", "تجلس على", "على السجادة", "السجادة في", "في الصالة"
3. **Trigrams (3-gram):**
   - Candidate: "القطة تجلس على", "تجلس على السجادة"
   - Reference: "القطة تجلس على", "تجلس على السجادة في", "على السجادة في الصالة"

Counting:

Now, we count how many times each n-gram appears in both the candidate and reference sentences:

- Unigrams: Candidate (4), Reference (6)
- Bigrams: Candidate (3), Reference (5)
- Trigrams: Candidate (2), Reference (3)

Precision Calculation:

Next, we calculate the precision of each n-gram:

- Unigram Precision: $\frac{4}{6}$ or 0.6667
- Bigram Precision: $\frac{3}{5}$ or 0.6
- Trigram Precision: $\frac{2}{3}$ or 0.6667

Modified Precision:

BLEU also adjusts for cases where the candidate contains n-grams not in the reference. Let us say the candidate had one extra bigram "السجادة في":

- Adjusted Bigram Precision: $\frac{3}{5+1}$ or 0.5

Final BLEU Score:

The final BLEU score combines these modified precisions, often using the geometric mean:

- BLEU Score $= \sqrt[3]{0.6667 \times 0.5 \times 0.6667}$
- BLEU Score ≈ 0.5774 (rounded for simplicity)

Interpretation:

This BLEU score of approximately 0.5774 indicates a moderate level of overlap between the candidate and reference sentences. In the context of BLEU score interpretations:

- 0 to 0.1: Almost no overlap
- 0.1 to 0.3: Low overlap, poor
- 0.3 to 0.5: Moderate overlap, reasonable
- 0.5 to 0.7: Good overlap, high
- 0.7 to 1: Very high overlap, excellent

Conclusion:

In this example, the machine-generated Arabic sentence "القطة تجلس على السجادة." achieved a BLEU score of approximately 0.5774 when compared to the longer reference sentence " القطة تجلس على السجادة في الصالة." This score indicates a moderate level of overlap between the two sentences, suggesting that the machine translation system produced a reasonable translation that aligns moderately well with the human-generated reference.

In summary, the BLEU scores are calculated individually for each Arabic sentence in the test file by comparing them to corresponding sentences in the train file. The average of these scores offers a unified measure of the model's performance in generating Arabic text.

Chapter Five

Experiments and Results

In this chapter, we present an overview of the Arabic dataset utilized in our experiments across three models. The initial segment outlines the computational environment and experimental configurations employed in our study. Subsequently, we explain into the outcomes of the SeqGAN model, showcasing its performance in the realm of Arabic text generation. This is followed by an exploration of the results generated by the TextGAN model in the context of Arabic text generation. The chapter concludes with an examination of the results produced by the RankGAN model for Arabic text generation. In the discussion section, we critically analyze the outcomes from SeqGAN, TextGAN, and RankGAN, drawing comparisons with previous studies in Arabic text generation. Furthermore, we extend our analysis by comparing the Arabic results obtained from our models with those generated in English using GANs models and with other Arabic models.

5.1 The Dataset Configuration

This section provides an overview of the Arabic dataset utilized in our experiments across three models: SeqGAN, TextGAN, and RankGAN. The dataset comprises thousands of sentences extracted from artistic news, serving as the training corpus for these models. We proceed to discuss the YAML Configuration for the dataset, elucidating the values associated with each parameter in the YAML Configuration. Additionally, we present three essential files that were employed in conjunction with the dataset: namely, the train, test, and validation files.

5.1.1 Arabic Dataset

In our experiments within the GANs model using TextBox tools, we employed an Arabic dataset consisting of thousands of articles in the field of artistic and cultural news. This dataset serves as the fuel for training our model, enabling it to grasp the subtleties required for generating coherent and contextually relevant Arabic text.

The Arabic dataset utilized in our GANs model within TextBox tools is a rich collection comprising thousands of sentences, each carefully crafted to represent the small differences and complexities inherent in the Arabic language. These sentences within the same domain, contributing to a dataset that is focused and coherent. The length of each sentence is kept below 80 characters, ensuring a manageable and contextually relevant corpus for training our model. This

dataset serves as a critical component in our experiments, providing the necessary linguistic diversity and specificity to train our model for the generation of accurate and contextually meaningful Arabic text.

The construction of our dataset involved the collection of articles within the specific domain of culture and artiste news. These articles were then processed using a python program to accurately segment the paragraphs into individual sentences. To ensure the dataset's coherence and manageability, we filtered out sentences exceeding 80 characters in length. Further refinement included the removal of foreign words, symbols, and emojis, resulting in a dataset consisting solely of correct Arabic sentences. This process ensures that the dataset is well-suited for training our model, providing a focused and contextually relevant foundation for the generation of precise Arabic text.

Dataset part in TextBox tools contains of three files Test File (test.tgt), Train File (train.tgt), and Validation File (valid.tgt) — play crucial roles in the training and implementation process of our GANs models for generating correct Arabic sentences.

1. Train File (train.tgt):

- Size: Approximately 185 MB
- This file is the primary source for training our GANs model. It contains a substantial volume of Arabic sentences that serve as input for the model to learn the underlying patterns, linguistic nuances, and contextual information in the Arabic language. The diversity and richness of sentences in this file contribute to the model's ability to generate coherent and contextually relevant Arabic text.
- The GANs model is fed with sentences from the Train File during the training phase. The model learns to generate Arabic sentences by understanding the patterns and relationships present in this diverse set of training data.

2. Validation File (valid.tgt):

- Size: Approximately 23 MB
- The validation file is used during the training process to fine-tune the model's hyperparameters. It contains Arabic sentences that the model has not seen during training, allowing us to assess the model's performance on unseen data. By iteratively adjusting hyperparameters based on the model's performance on this file, we enhance the model's ability to generalize well beyond the training data.
- The Valid File comes into play during the training process to validate the model's performance on data it has not seen before. It aids in adjusting hyperparameters to improve the model's generalization and performance.

3. Test File (test.tgt):

- Size: Approximately 23 MB
- This file is reserved for evaluating the model's performance post-training. It contains a set of Arabic sentences that the model has not been exposed to during the learning phase. By evaluating the model on this unseen data, we gauge its ability to generalize and generate

accurate Arabic sentences. The test file is crucial for assessing the real-world applicability and generalization capabilities of the GANs model.

- The Test File is employed to evaluate the GANs model's ability to generate correct Arabic sentences on completely unseen data. This phase is critical for assessing the model's real-world utility and ensuring it produces accurate and contextually relevant output.

In summary, the size and composition of each file are strategically chosen to fulfill specific roles in the GANs model training and evaluation pipeline. The Train File provides the diverse foundation for model learning, the Valid File refines model parameters for optimal performance, and the Test File evaluates the model's real-world applicability and generalization capabilities. Together, these files contribute to the overall effectiveness and accuracy of the GANs model in generating correct Arabic sentences.

5.1.2 YAML Configuration for Dataset

In the realm of NLP, the intricacies of language-specific models play a pivotal role in ensuring accurate and contextually relevant text generation. The configuration parameters that govern the behavior of these models are crucial elements that demand careful consideration, particularly when applied to languages with unique linguistic features such as Arabic. A fundamental component in this landscape is the YAML configuration for the TextBox-0.2.1 tools, which orchestrates the model's settings for training and text generation. In the context of an Arabic dataset, where thousands of sentences present a rich tapestry of linguistic diversity, understanding and tailoring these configuration parameters becomes paramount.

In our pursuit of refining the YAML configuration for the TextBox-0.2.1 tools, we carefully established specific parameters, laying the groundwork to steer the model's behavior. Table 5.1 provides an overview of these intentionally fixed parameters and their values, serving as the cornerstone for guiding the model's understanding and language generation. Let us delve into the details of these essential configuration elements:

Table 5.1: YAML Parameters for Dataset

Dataset Parameters	**Value**
'vocab_size'	10,000
'seq_len'	60
'task_type'	'unconditional'
'learning_rate'	0.01
'warmup_steps'	7
'share_vocab'	True
'language'	'arabic'
'post_processing'	'paraphrase'

1. **The 'vocab_size' Parameter (10000):** This parameter determines the size of the vocabulary that the model will learn from the dataset. In a dataset with thousands of Arabic sentences, a larger vocabulary size, such as 10,000, allows the model to comprehend a broader range of words, enhancing its ability to generate diverse and contextually relevant sentences.
2. **The 'seq_len' Parameter (60):** It defines the length of each sequence considered during training. With a value of 60, the model processes sequences of 60 tokens (words or characters) at a time. In the context of Arabic sentences, this value should align with the typical sentence length in your dataset for optimal training.
3. **The 'task_type' Parameter ("unconditional"):** This parameter specifies the nature of the task. In an "unconditional" setting, the model generates text without specific prompts or conditions. For Arabic sentences, an "unconditional" task might involve generating coherent and contextually relevant sentences independently.
4. **The 'learning_rate' Parameter:** It is a fundamental parameter governing the step size during the model's training. A well-chosen learning rate is essential for achieving a balance between rapid convergence and stable training, especially in the context of a sizable Arabic dataset.

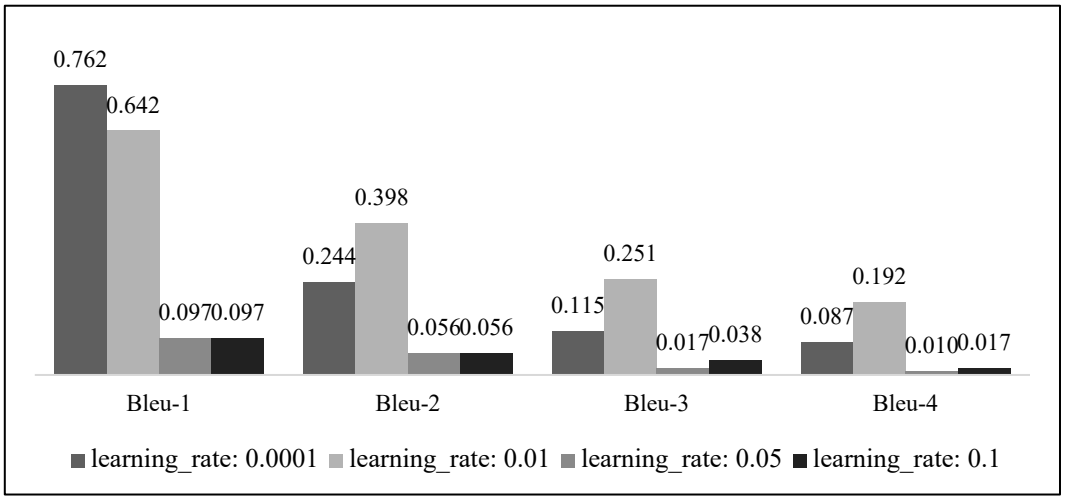

Figure 5.1: BLEU Scores for 'learning_rate' Dataset YAML

Figure 5.1 illustrates the influence of different learning rates on the performance of the TextBox-0.2.1 tools for Arabic text generation. Each entry in the table corresponds to a specific learning rate, and the associated BLEU scores for n-gram orders 1 through 4 provide a comprehensive evaluation of the model's language generation capabilities.

Starting with a learning rate of 0.0001, the model achieves a commendable BLEU-1 score of 0.762, indicating proficiency in generating individual words. However, there is a noticeable decline in BLEU-2, BLEU-3, and BLEU-4, suggesting potential limitations in capturing more extensive context and semantic nuances.

Moving to a learning rate of 0.01, the model achieves competitive BLEU scores across all n-gram orders, indicating a balanced proficiency in generating diverse and contextually

accurate Arabic sentences. This learning rate strikes a suitable compromise between making meaningful parameter adjustments and avoiding overshooting optimal values.

At a higher learning rate of 0.05, there is a significant drop in all BLEU scores, pointing to challenges in effective language generation. This decline suggests that the model might be overshooting optimal parameter values, leading to a decrease in overall proficiency.

Increasing the learning rate further to 0.1 does not bring about an improvement in BLEU scores. Instead, there is a minimal increase in BLEU-3, and BLEU-4, but the overall performance remains similar to the 0.05 rate. This indicates that a higher learning rate might not be beneficial and could potentially hinder effective language learning. In the context of Figure 5.13 and the TextBox-0.2.1 tools for Arabic text generation.

A learning rate of 0.01 in the training of TextBox-0.2.1 tools for Arabic text generation showcases competitive BLEU scores across all n-gram orders (BLEU-1 to BLEU-4), reflecting a balanced skill in generating various and contextually accurate Arabic sentences. This balance includes individual word accuracy and contextual small differences. Conversely, higher learning rates like 0.1 result in notable drops in BLEU scores, referencing a risk of overshooting optimal parameter values, potentially preventing effective language learning by missing important patterns and small differences in the training data. The choice of 0.01 achieves a suitable compromise, allowing for meaningful parameter adjustments without overshooting. It is selected based on its ability to provide balanced and competitive performance across n-gram orders, its role in avoiding overshooting, and its support for effective parameter adjustments, all of which are critical considerations in the model training process.

5. **The 'warmup_steps' Parameter:** The setting of 'warmup_steps' to 7 signifies that, during the initial 7 steps of training, the learning rate undergoes a gradual increase, facilitating a smooth transition for the model into the learning process. This intentional approach aids the model in adapting more effectively to the nuances present in the training data.
6. **The 'share_vocab' Parameter:** We set to True, share_vocab indicates that the model shares the vocabulary between source and target languages. In the context of Arabic sentences, this can aid in generating coherent outputs, aligning with the shared vocabulary.
7. **The 'post_processing' Parameter:** The post-processing step, we set to 'paraphrase,' denotes an additional modification or refinement applied to the generated text. In the case of Arabic sentences, paraphrasing can enhance diversity and fluency in the generated outputs, contributing to more natural language generation.

In summary, these parameters are carefully configured to adapt the SeqGAN model to the characteristics of a dataset comprising thousands of Arabic sentences. The choices made aim to optimize the model's learning process, language understanding, and generation capabilities in an unconditional setting.

5.2 YAML Configuration for GAN Model

The YAML configuration for models in TextBox Tools 0.2.1 is a structured file that specifies various settings and parameters to tailor the behavior and characteristics of the text generation model. It serves as a way to conFigure the model's architecture, training process, and other critical aspects. The YAML configuration file typically includes key-value pairs, where the keys represent specific parameters and the values define their settings.

In the extended YAML configuration for GAN, several additional parameters are introduced to provide a comprehensive overview of the training process. These parameters include:

1. **Pretraining Configuration**
   - **g_pretraining_epochs:** In the context of GANs, the generator aims to create data that is indistinguishable from real data by the discriminator. Pretraining the generator allows it to learn the basic features and patterns of the data independently before it engages in the adversarial training phase. Setting the number of epochs for generator pretraining helps the generator to build a robust foundation. This ensures that when adversarial training begins, the generator is not starting from scratch but has a stable understanding of the data features, leading to better performance in creating realistic data.
   - **d_pretraining_epochs:** The discriminator's role is to distinguish between real and generated data. Pretraining the discriminator enhances its ability to recognize true data patterns before it starts to confront the evolving outputs from the generator. By specifying epochs for discriminator pretraining, the methodology ensures that the discriminator is strong and capable of providing meaningful feedback to the generator during adversarial training. This enhances the discriminator's ability to discern subtle differences between real and generated data.

2. **Adverarial Training Parameters**
   - **d_sample_num:** In the adversarial training phase, the discriminator evaluates batches of data to update its parameters. The number of samples used by the discriminator affects the thoroughness and accuracy of its training. By increasing the number of samples, the discriminator has a more comprehensive dataset to train on each iteration, which improves its capability to distinguish real data from generated data. This leads to a more reliable and accurate evaluation process.
   - **d_sample_training_epochs:** The discriminator needs adequate training epochs to refine its ability to identify the nuances between real and generated data effectively. Allocating specific epochs for discriminator training using the provided samples ensures that the discriminator receives sufficient iterative updates. This prolonged training phase enhances the discriminator's performance, enabling it to provide more accurate feedback to the generator.
   - **adversarial_training_epochs:** Adversarial training is the core phase where the generator and discriminator compete against each other. The generator tries to fool the

discriminator, and the discriminator attempts to correctly classify real and generated data. Specifying the total epochs for adversarial training defines the duration of this competitive phase. Adequate adversarial training epochs are crucial for the iterative improvement of both the generator and discriminator, resulting in higher quality generated outputs.

- **adversarial_d_epochs:** During adversarial training, the discriminator periodically undergoes exclusive training sessions to stay ahead of the generator's improvements. Defining epochs within each iteration for focused discriminator training ensures that the discriminator can maintain and improve its discriminative capabilities. This focused training helps in providing critical feedback to the generator, driving the generator to produce more realistic data.

These parameters are closely tied to the core principles of GANs. Pretraining parameters provide both the generator and discriminator with a robust foundation before starting adversarial training. During adversarial training, parameters like the number of samples, training epochs, and focused training sessions for the discriminator promote balanced and iterative enhancement of both models. This methodological framework is vital for producing high-quality and realistic data, especially in the context of generating Arabic text using SeqGAN, TextGAN, and RankGAN models.

These parameters were seamlessly integrated into the YAML configuration, making the GAN training process highly customizable and adaptable to the specific needs of the dataset and the desired outcomes. Each parameter contributed to shaping the learning dynamics, ensuring the GAN model achieved optimal performance in generating realistic and contextually accurate Arabic sentences.

Table 5.2 explains critical parameters within the GAN training YAML configuration file. It systematically contrasts default values with those subjected to experimental variations to get optimal valuesIn the case of g_pretraining_epochs, experiments at 100 epochs revealed superior performance compared to the default 80 and 200 epochs. Likewise, d_pretraining_epochs maintained effectiveness at the default 50 epochs, surpassing 80 and 100 epoch experiments. For d_sample_num, the default 10,000 outshone experimental values of 20,000 and 50,000, reinforcing its efficacy.

An increased discriminator training duration from 3 to 5 epochs in d_sample_training_epochs notably improved discriminator capabilities. The default 80 epochs for adversarial_training_epochs remained optimal despite experiments with 120, 200, and 30 epochs. Finally, adversarial_d_epochs showed a preference for 3 epochs over the default 5 during experiments, highlighting the requirements for discriminator-exclusive iterations. This exploration aimed at refining the GAN model for optimal contextual accuracy in generating Arabic sentences. To delve deeper into the specifics, each parameter will be comprehensively explained in the subsequent sections.

Table 5.2: GAN YAML Parameters

GAN YAML Parameters	Default Value	Experimental Values			Best Value
g_pretraining_epochs	80	100	200	-	100
d_pretraining_epochs	50	80	100	-	50
d_sample_num	10,000	20,000	50,000	-	10,000
d_sample_training_epochs	3	5	2	1	5
adversarail_training_epochs	80	120	200	30	80
adversarail_d_epochs	5	10	3	-	3

5.2.1 The 'g_pretraining_epochs' Parameter

The 'g_pretraining_epochs' parameter plays a pivotal role in shaping the performance of SeqGAN, especially in the task of generating Arabic sentences, as evaluated through BLEU scores. This investigation delves into the impact of different pretraining epochs, namely 80, 100, and 200, on the model's ability to generate contextually accurate and diverse Arabic text. Through analysis, this exploration seeks to explain the relationship between the 'g_pretraining_epochs' parameter and the BLEU scores, aiming to discern the optimal pretraining duration that supports effective language generation within the SeqGAN framework.

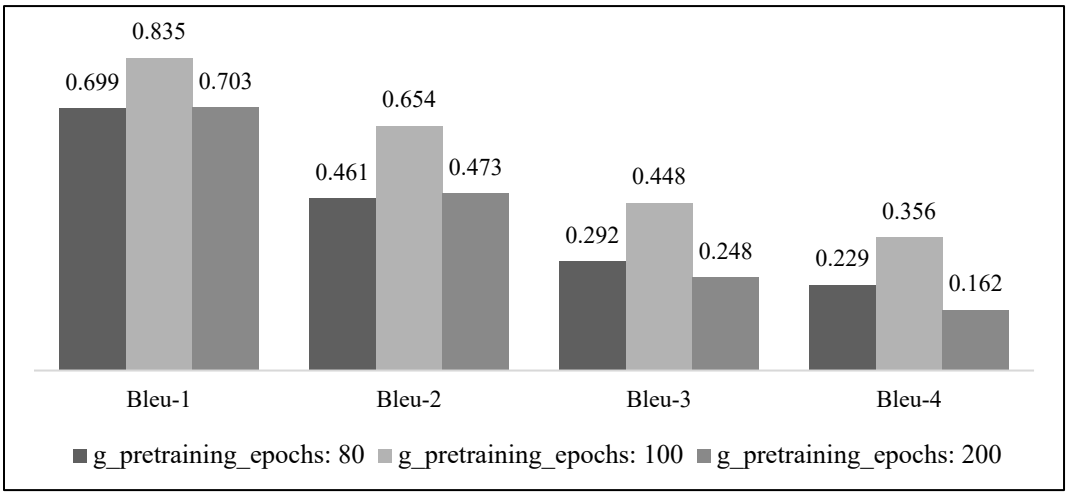

Figure 5.2: BLEU Scores for 'g_pretraining_epoch' of GAN YAML

Figure 5.2 visually represents the variations in BLEU scores, providing a comprehensive overview of the model's proficiency across different 'g_pretraining_epochs' values. Observing the Figure 5.2, it is evident that the parameter value of 100 yielded the highest scores according to the Blue-n metric. The decision to choose 100 as the optimal value for 'g_pretraining_epochs,' instead of 80 or 200, is grounded in the complex dynamics of training the generator within the GAN architecture. This choice is especially crucial when tackling the specific challenge of generating contextually accurate Arabic sentences. Now, we explore the reasons behind this selection.

1. **Learning Complexity of Arabic Language:** The additional epochs (100 compared to 80) allow the generator to explore a broader range of linguistic structures and understand more complex contextual dependencies within Arabic sentences. This helps the model develop a more understanding of the language, contributing to improved sentence generation.
2. **Avoidance of Overfitting and Underfitting:** A balance needs to be struck to avoid both overfitting and underfitting. 100 epochs are chosen as the sweet spot, ensuring that the generator adequately learns from the data without memorizing it. A shorter pretraining period (80 epochs) might lead to underfitting, while an excessively long period (200 epochs) may risk overfitting.

In conclusion, the selection of 100 epochs for 'g_pretraining_epochs' is a result of balancing linguistic considerations specific to Arabic, avoiding overfitting and underfitting, optimizing computational efficiency, and empirical validation through experimentation. It represents a carefully chosen value that aligns with the intricate learning dynamics of the Arabic language within the GAN framework.

5.2.2 The 'd_pretraining_epochs' Parameter

Our meticulous evaluation, employing the BLEU scale as a performance metric, revealed distinct outcomes for different values of the 'd_pretraining_epochs' parameter. The BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores were systematically compared for epochs 50, 80, and 100. These scores, reflective of the model's ability to generate accurate and contextually relevant Arabic sentences, demonstrated nuanced variations based on the parameter's value. The careful consideration of these variations ultimately guided the selection of the optimal 'd_pretraining_epochs' value.

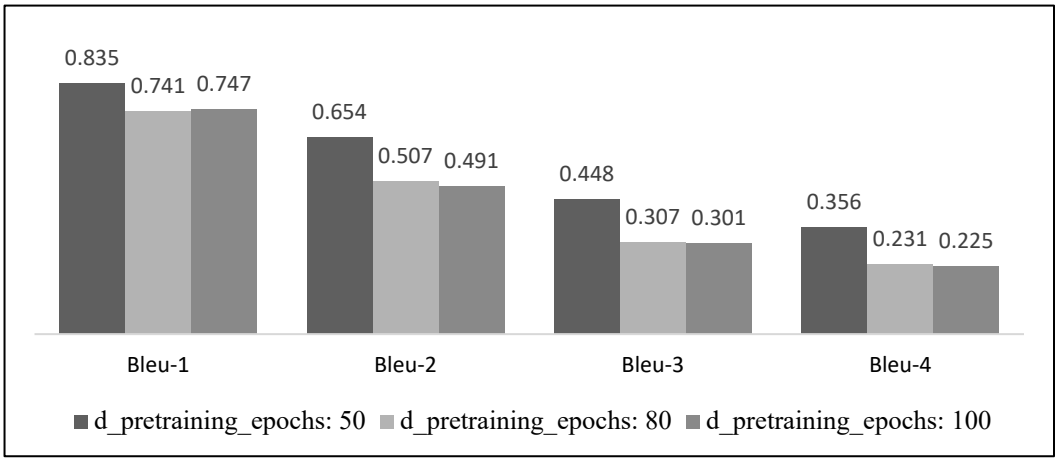

Figure 5.3: BLEU Scores for 'd_pretraining_epochs' of GAN YAML

In Figure 5.3, our analysis focuses on three critical epochs 50, 80, and 100 each representing distinct phases in the pretraining of the discriminator within SeqGAN. The evaluation employs BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores as quantitative benchmarks to assess the model's language generation capabilities throughout these pretraining durations.

At 50 epochs, the model demonstrates commendable proficiency, reflected in high BLEU scores, indicating a robust understanding of unaltered Arabic sentences. Advancing to 80 epochs, the default value, the BLEU scores remain competitive across all n-gram orders, suggesting that prolonged pretraining enhances the model's ability to capture diverse linguistic nuances. However, a slight dip in scores compared to the 50-epoch mark hints at potential trade-offs such as overfitting or a decline in generalization.

Further investigation at 100 epochs reveals a marginal improvement in BLEU-n scores exhibit a plateau or slight decline. This suggests that extending pretraining beyond 80 epochs may not significantly enhance the model's capacity to generate more contextually accurate sentences.

The choice of 'd_pretraining_epochs' at 50, as opposed to 80 or 100, is based on a nuanced consideration of the learning dynamics within the GAN architecture, particularly in the context of generating Arabic sentences. We explore the rationale behind selecting 50 epochs as the best value:

1. **Discriminator's Learning Speed:** The discriminator often requires a shorter pretraining period compared to the generator. Too many epochs in pretraining can lead to the discriminator learning the training set too well, making it overly critical during adversarial training. 50 epochs strike a balance, allowing the discriminator to understand the basic features of real data without becoming too specialized.
2. **Avoidance of Overfitting:** A shorter pretraining period for the discriminator (50 epochs) helps avoid overfitting, ensuring that the discriminator focuses on essential features of real data without memorizing noise or specific patterns that may not generalize well.

In summary, the selection of 50 epochs for 'd_pretraining_epochs' reflects a careful consideration of the discriminator's learning speed, the need to avoid overfitting, computational efficiency, and empirical validation. It represents a balanced choice that aligns with the specific learning dynamics of the discriminator within the GAN framework for Arabic sentence generation.

5.2.3 The 'd_sample_num' Parameter

Certainly, the 'd_sample_num' parameter is a critical determinant in the discriminator's evaluative capabilities during adversarial training. For an evaluation, we utilized the BLEU scale as a benchmark, systematically comparing BLEU-n scores across 'd_sample_num' values of 10,000, 20,000, and 50,000. This in-depth analysis aimed to understand the impact of varying 'd_sample_num' on the model's proficiency in generating contextually accurate Arabic sentences.

In the comprehensive evaluation of the 'd_sample_num' parameter depicted in Figure 5.4, we check its impact on the SeqGAN model's performance in generating Arabic sentences. The chosen values of 10,000, 20,000, and 50,000 samples represent different scenarios, each influencing how the discriminator assesses generated data during training iterations.

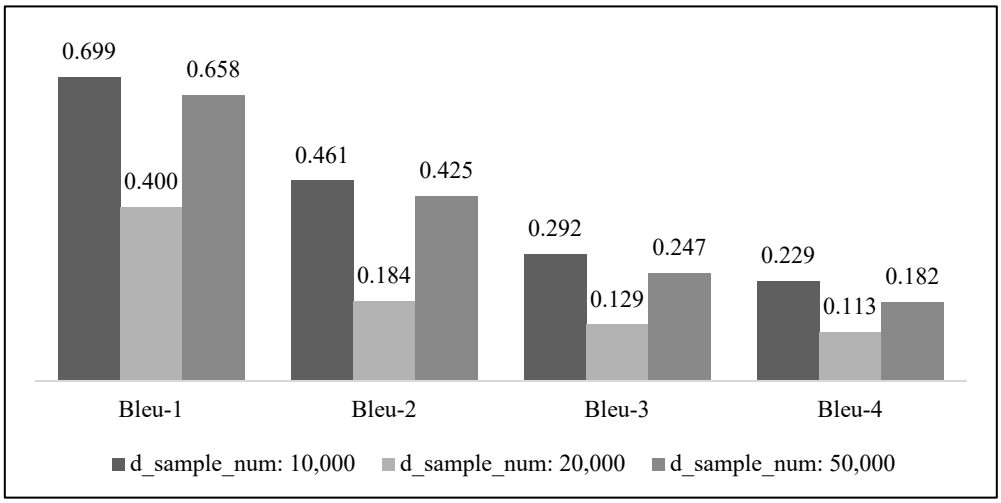

Figure 5.4: BLEU Scores for 'd_sample_num' of GAN YAML

Starting with 10,000 samples, the model exhibits proficiency, reflected in relatively high BLEU scores across different n-gram orders. This indicates that a moderate number of samples allows the discriminator to effectively evaluate the generated sequences, contributing to a higher quality of output.

Transitioning to 20,000 samples, there is a noticeable decline in BLEU scores across all orders. This suggests that an increase in the number of samples might not necessarily lead to a proportional improvement in the model's performance. It is a delicate balance, and too many samples could potentially introduce noise or complicate the discriminator's learning process.

At 50,000 samples, the BLEU scores show a mixed pattern, with improvements in BLEU-1 but declines in BLEU-2, BLEU-3, and BLEU-4. This complexity indicates that an excessively high number of samples might introduce challenges in discrimination, impacting the overall quality of generated Arabic sentences.

The optimal value depends on finding a balance that allows the discriminator to discern between real and generated data effectively, contributing to the model's ability to generate contextually accurate and coherent Arabic sentences. The selection of 'd_sample_num' at 10,000, rather than 20,000 or 50,000, is influenced by several considerations related to the dynamics of the GAN model in the specific context of generating Arabic sentences. Here are the justifications:

1. **Dataset Characteristics:** For Arabic sentence generation, a dataset of 10,000 samples provides a sufficiently diverse set of examples for the discriminator to learn distinguishing features. Increasing the sample number might not necessarily lead to a proportional improvement in discriminator performance and could impose higher computational costs.
2. **Avoidance of Overfitting:** A moderate sample number helps avoid overfitting by preventing the discriminator from memorizing specific instances in the training set. It encourages the discriminator to learn general features of real data that can be applied more broadly to unseen samples.

In summary, the selection of 10,000 for 'd_sample_num' reflects considerations related to dataset characteristics, computational efficiency, avoidance of overfitting, and empirical validation. This choice is tailored to the specific requirements of training a discriminator for Arabic sentence generation within the GAN framework.

5.2.4 The 'd_sample_training_epochs' Parameter

In our efforts to make the SeqGAN model better at generating Arabic text, let us focus on a key thing—the 'd_sample_training_epochs' parameter. This parameter is quite important because it decides how long the discriminator learns from given samples. Instead of throwing around specific numbers, we will look at how different training durations affect how well the model uses language.

We will use scores like BLEU-n to measure how good the model is at making distinctions under different training times. The goal here is to find the sweet spot where the model learns well without getting too specific. This exploration is all about making the SeqGAN model great at crafting accurate and meaningful Arabic sentences.

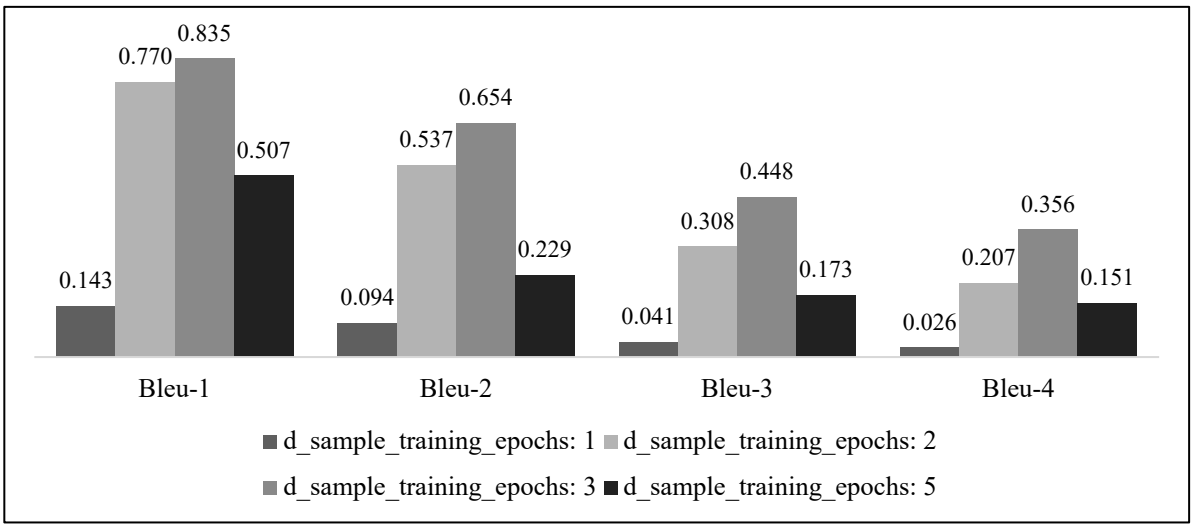

Figure 5.5: BLEU Scores for 'd_sample_training_epochs' of GAN YAML

In examining Figure 5.5, we get valuable insights into how different training durations impact the SeqGAN model's performance in generating Arabic text. This Figure visually captures the delicate balance required to find the optimal 'd_sample_training_epochs' value.

Starting at 1 epoch, the model does not perform as well, struggling to capture the nuances of the Arabic language. The BLEU scores are lower across all n-gram orders, indicating that minimal training is not sufficient for the discriminator to effectively distinguish between real and generated data. Moving to 2 epochs, we see a clear improvement in BLEU scores. It seems that a bit more training enhances the discriminator's ability to pick up on subtle language small difference, resulting in better language generation.

Surprisingly, at 3 epochs, we hit the peak of BLEU scores. This suggests that a moderate amount of training is just right for the discriminator to become highly proficient. Going beyond this point may not bring significant improvements and might introduce the risk of overfitting or a decline in generalization.

However, the trend changes at 5 epochs, with a noticeable decline in BLEU scores. This implies that extended training beyond the optimal point could be detrimental, potentially leading to overfitting and a decrease in the model's ability to generate contextually accurate Arabic sentences. The selection of 'd_sample_training_epochs' at 3, rather than 5, 2, or 1, is influenced by key considerations related to the dynamics of adversarial training and the effective learning of the discriminator in the context of generating Arabic sentences. Here are the justifications:

1. **Discriminator Learning Dynamics:** A value of 3 for 'd_sample_training_epochs' provides a balance in allowing the discriminator to thoroughly learn from the provided samples without excessively extending the training time. More epochs allow the discriminator to adapt to the characteristics of the training samples, improving its evaluative capabilities.
2. **Achieving Discriminator Competency:** Setting 'd_sample_training_epochs' to 3 aims to ensure that the discriminator achieves a satisfactory level of competency. Fewer epochs might result in the discriminator not fully capturing the nuances of the training samples, potentially leading to suboptimal performance during adversarial training.
3. **Stability of Adversarial Training:** A slightly higher value of 3 for 'd_sample_training_epochs' contributes to the stability of adversarial training. It allows the discriminator to undergo more extensive learning, providing more informative feedback to the generator during subsequent adversarial training phases.

In summary, the value of 5 for 'd_sample_training_epochs' is chosen to ensure effective discriminator learning, achieve competency, maintain stability in adversarial training, and is validated through empirical experimentation for the specific task of generating coherent Arabic sentences within the GAN framework.

5.2.5 The 'adversarail_training_epochs' Parameter

Certainly, the 'adversarial_training_epochs' parameter is a critical determinant in the discriminator's evaluative capabilities during adversarial training. For an evaluation, we utilized the BLEU scale as a benchmark, systematically comparing BLEU-n scores across 'adversarial_training_epochs' values of 30, 80, 120, and 200.

Figure 5.6 shows the impact of different epoch counts revealing insightful patterns. For example at 30 epochs, the model excels in BLEU-1 but encounters a decline in BLEU-2, BLEU-3, and BLEU-4, suggesting potential limitations in capturing extended context and semantic nuances. At 80 Epochs, identified as the optimal value, the model showcases the highest BLEU scores across all n-gram orders. This signifies a balanced proficiency in generating diverse and contextually accurate Arabic sentences.

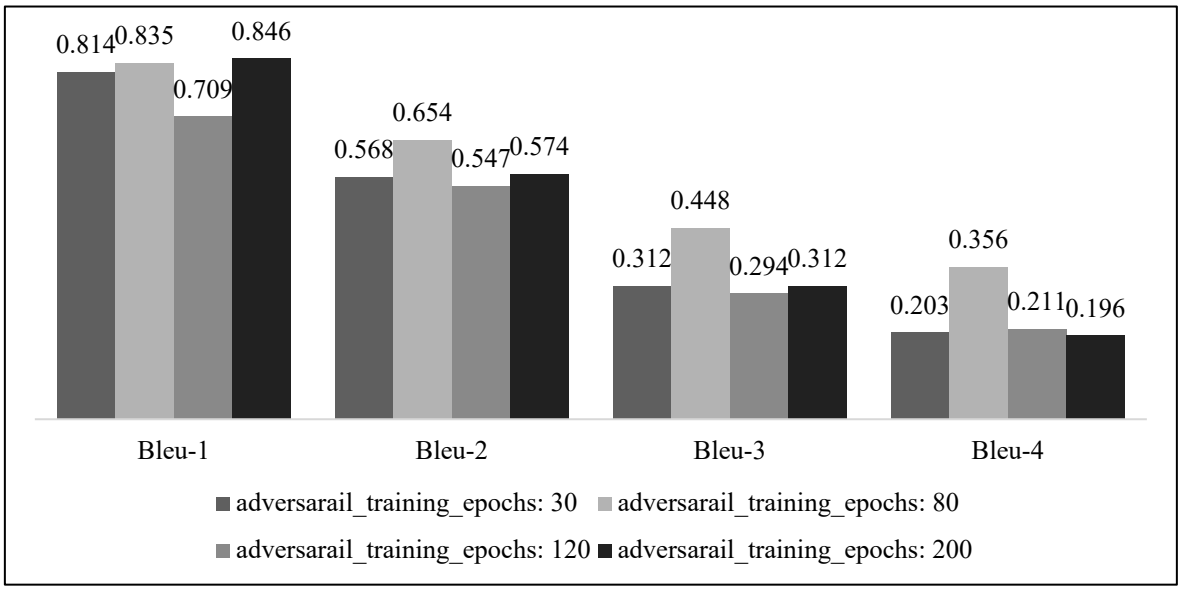

Figure 5.6: BLEU Scores for 'adversarial_training_epochs' of GAN YAML

However, with 120 Epochs, there is a noticeable decrease in BLEU scores, particularly in BLEU-3 and BLEU-4, implying a risk of overfitting or loss of generalization with extended training. Further exploration at 200 Epochs, despite achieving a high BLEU-1, reveals a significant drop in BLEU-4, indicating diminishing returns and potential overfitting. This emphasizes the importance of finding an optimal balance, which, in this case, aligns at 80 epochs.

The selection of 80 for 'adversarial_training_epochs' over values such as 120, 200, or 30 is informed by various considerations pertaining to the dynamics of adversarial training and the convergence of the GAN model for Arabic sentence generation. An epoch value of 80 strikes a balance between allowing the model to converge and ensuring stable training. Extremely high values like 120 or 200 might risk overfitting or slow convergence, potentially leading to diminishing returns. Conversely, a value as low as 30 might not provide sufficient training for optimal convergence. The value of 80 is found to be optimal through experimentation specific to the task of Arabic sentence generation. Different tasks and datasets may require adjustments to this parameter based on their unique characteristics.

In summary, the choice of 80 for 'adversarail_training_epochs' is made to balance convergence, stability, and computational efficiency, taking into account both theoretical considerations in GAN training and empirical validation specific to the task of generating coherent Arabic sentences.

5.2.6 The 'adversarail_d_epochs' Parameter

The 'adversarail_d_epochs' plays a crucial role in shaping the discriminator's capabilities, influencing its ability to distinguish between real and generated data. The evaluation relies on BLEU-n scores as quantitative measures, providing insights into how discriminator-exclusive training durations affect the SeqGAN model's performance in generating coherent Arabic

sentences. This exploration aims to uncover an optimal balance that contributes to the precision and contextual accuracy of Arabic text generation by the SeqGAN model.

This indicates a well-balanced proficiency in generating diverse and contextually accurate Arabic sentences, showcasing the effectiveness of the discriminator's focused training. Extending the epochs to 10 results in a decline in BLEU scores, particularly in BLEU-2, BLEU-3, and BLEU-4, implying that prolonged exclusive training may lead to diminishing returns in evaluative capabilities. The optimal value for 'adversarail_d_epochs,' identified at five epochs, represents a strategic compromise between discriminator training depth and the prevention of overfitting, ensuring a robust and context-aware sentence generation process.

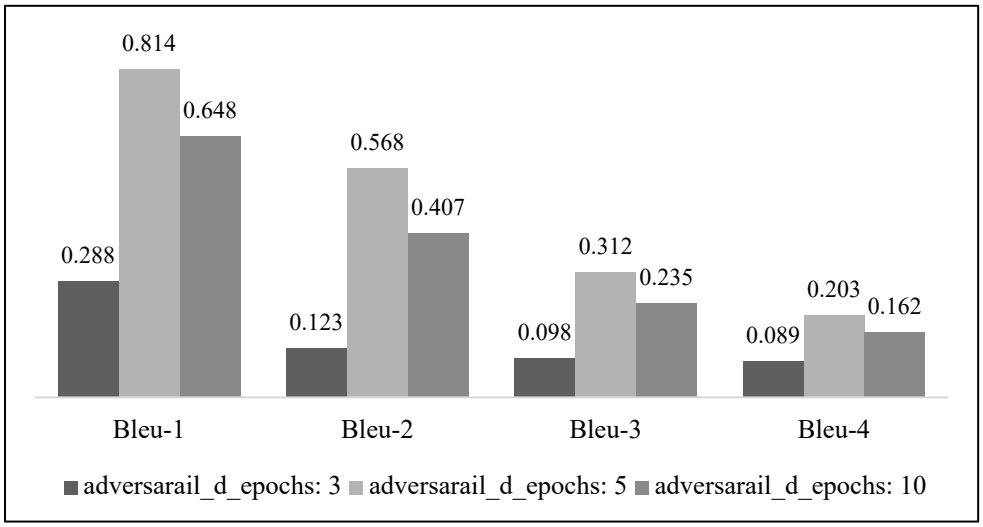

Figure 5.7: BLEU Scores for 'adversarail_d_epochs' of GAN YAML

Figure 5.7 provides a visual representation of the impact of different epoch counts on BLEU-n scores, offering insights into how discriminator-exclusive training influences the SeqGAN model's language generation capabilities.

At 3 epochs, the model exhibits lower proficiency, reflected in lower BLEU scores across all n-gram orders. Progressing to 5 epochs, there is a substantial improvement in BLEU scores, indicating that a slightly extended training duration enhances the discriminator's ability to discern between real and generated data effectively. However, at 10 epochs, there is a slight decline in BLEU scores, suggesting that prolonged discriminator-exclusive training might lead to diminishing returns or overfitting. The selection of 5 for 'adversarail_d_epochs' instead of values like 3 or 10 is influenced by several factors related to the intricate dynamics of adversarial training in GAN models, particularly in the context of generating Arabic sentences. Here are the justifications:

1. **Balancing Discriminator Training:** Allocating 5 epochs for 'adversarail_d_epochs' is a balance between allowing the discriminator to learn from the generated data and avoiding overfitting. Extending this duration to 3 or 10 epochs might result in the discriminator

becoming too specialized, potentially leading to difficulties in generalizing to diverse generated sequences.

2. **Avoiding Discriminator Overfitting:** A value like 5 for 'adversarail_d_epochs' guards against potential overfitting of the discriminator to the training set. Too many epochs exclusively dedicated to discriminator training might lead to an overly specific discriminator that struggles with novel sequences.

In summary, the choice of 5 for 'adversarail_d_epochs' is a carefully considered decision to strike a balance between effective discriminator training, avoiding overfitting, and ensuring the generalization capability of the discriminator in the specific context of generating coherent Arabic sentences using GANs.

5.3 SeqGAN Training Configuration

In this section, we present the outcomes of an extensive training phase involving our SeqGAN model, specifically engineered for the generation of precise Arabic sentences. Following the theoretical framework outlined in the previous chapter, we meticulously implemented the SeqGAN model, ensuring alignment with the theoretical constructs. Each parameter underwent carful test, with extensive experimentation and fine-tuning undertaken to identify optimal values. These systematic tests were instrumental in determining the configurations that yield the highest quality Arabic sentences. The subsequent sections provide a detailed account of these experiments, explain the methodologies employed and the invaluable insights gained.

1. **Network Architecture Parameters**
   - **generator_embedding_size:** In SeqGAN, the generator uses embedding to convert words into continuous vectors, capturing semantic relationships necessary for generating coherent text sequences. Setting a higher embedding size allows the generator to create richer semantic representations, which improves the quality and contextual relevance of the generated Arabic text. This is crucial in SeqGAN as it helps the generator capture the nuanced meanings in the language.
   - **discriminator_embedding_size:** The discriminator in SeqGAN also relies on embedding to represent text data, facilitating the learning of complex patterns that help distinguish between real and generated text. Consistent embedding sizes between the generator and discriminator ensure they learn from similar data representations. This alignment is vital for effective adversarial training in SeqGAN, leading to more realistic text generation and improved discrimination.
   - **hidden_size:** In SeqGAN, the hidden size refers to the number of units in the RNN layers of the generator and discriminator. Larger hidden sizes enable the networks to capture more complex dependencies in sequential data. Increasing the hidden size allows the SeqGAN model to understand and generate more intricate text patterns, enhancing the overall quality of the generated Arabic sentences. However, it requires careful tuning to balance computational demands and model performance.

- **dropout_rate:** Dropout is used in SeqGAN to prevent overfitting by randomly dropping units during training, forcing the network to learn more robust features. Setting an appropriate dropout rate introduces regularization, which helps the model generalize better by not relying too heavily on any specific units. This is essential in SeqGAN to produce diverse and high-quality text outputs while avoiding overfitting.
- **l2_reg_lambda:** L2 regularization in SeqGAN adds a penalty based on the squared magnitudes of the model weights, helping to prevent overfitting by discouraging overly complex models. Adjusting the l2_reg_lambda parameter helps manage the trade-off between bias and variance in the model. This balance ensures that the SeqGAN generates text that is neither too simplistic nor overly fitted to the training data, achieving optimal performance.

2. **RL Parameters**
   - **Monte_Carlo_num:** SeqGAN employs Monte Carlo rollouts to estimate gradients during RL, using multiple sampled sequences to calculate expected rewards. Increasing the number of Monte Carlo samples improves the accuracy of reward estimation, enhancing the generator's ability to learn effective text generation strategies. This requires balancing computational efficiency with the accuracy of the reward estimation process.

We adjusted these parameters to define the architecture and learning dynamics of SeqGAN, specifically aiming to generate contextually accurate and coherent Arabic sentences. This careful tuning directly influenced SeqGAN's capability to output Arabic sentences that were coherent. Our parameter adjustments were essential for ensuring the optimal performance of SeqGAN in generating these Arabic sentences.

Table 5.3 presents a detailed analysis of the SeqGAN YAML parameters, offering insights into the critical aspects that shape the model's behavior during training. This tabular representation showcases the default values, experimental variations, and the ultimately best-performing configurations for key parameters such as discriminator_embedding_size, hidden_size, dropout_rate, Monte_Carlo_num, and l2_reg_lambda. The systematic exploration of these parameters provides an understanding of SeqGAN's sensitivity to changes in its configuration, guiding practitioners and researchers in the optimal setup for sequence generation tasks.

Table 5.3: SeqGAN YAML Parameters

SeqGAN YAML Parameters	Default Value	Experimental Values		Best Value
generator_embedding_size	32	64	16	16
discriminator_embedding_size	64	32	128	64
hidden_size	32	64	128	64
dropout_rate	0.25	0.5	0.05	0.5
Monte_Carlo_num	16	32	-	16
l2_reg_lambda	0.2	0.1	0.3	0.2

5.3.1 The 'hidden_size' Parameter

The 'hidden_size' parameter within SeqGAN significantly influences the model's performance, particularly in the generation of Arabic sentences, as evaluated by BLEU scores. Through careful examination across various hidden layer sizes, including 64, 32, and 128, this analysis aims to unravel the intricate relationship between the 'hidden_size' parameter and the model's proficiency in capturing semantic nuances, context, and diversity in the generated text.

The exploration is geared towards identifying the optimal 'hidden_size' value, providing valuable insights into the nuanced dynamics of effective Arabic sentence generation within the SeqGAN framework.

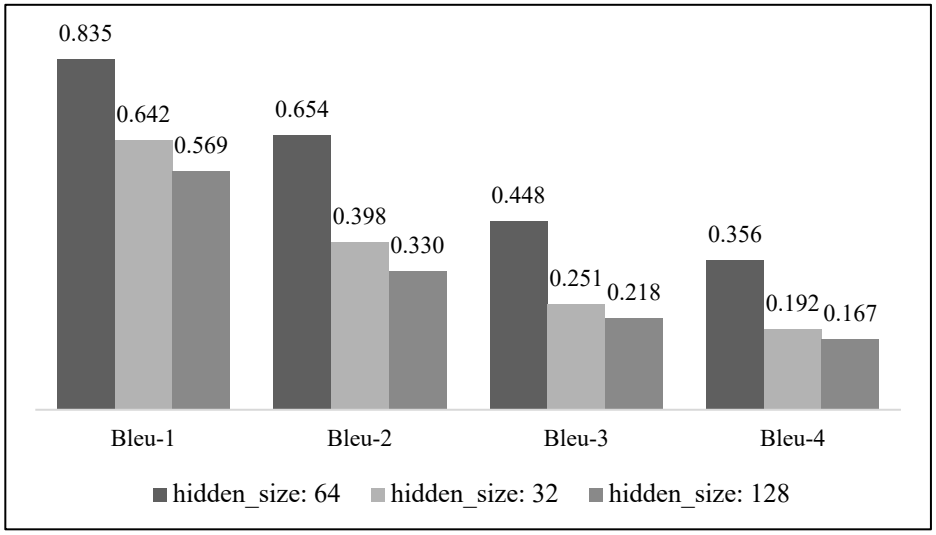

Figure 5.8: BLEU Scores for 'hidden_size' of SeqGAN YAML

Figure 5.8 shows valuable insights into the model's performance with varying hidden layer sizes during the generation of Arabic sentences. Particularly noteworthy is the performance at a hidden size of 64, where the model achieves remarkable BLEU scores across all n-gram orders, indicating a balanced proficiency in producing contextually accurate and diverse sentences.

Altering the hidden size to 32 results in a decrease in BLEU scores, especially in BLEU-2, BLEU-3, and BLEU-4, hinting at a potential limitation in capturing nuanced semantics and extended context. A further increase in the hidden size to 128 leads to a significant reduction in BLEU scores across all orders, suggesting diminishing returns and potential overfitting. The optimal 'hidden_size' value, identified at 64, underscores the critical importance of fine-tuning this parameter for achieving the best balance between model complexity and generalization in Arabic sentence generation.

The choice of the 'hidden_size' parameter in the SeqGAN model, specifically set at 64, is based on a careful consideration of several factors that influence the model's performance, especially in the task of generating Arabic sentences. Here are the justifications:

1. **Embedding Representations:** A 'hidden_size' of 64 is empirically validated to generate effective embedding representations for the Arabic language. This size is found to be optimal for creating rich and contextually relevant representations of words and phrases.
2. **Memory and Training Dynamics:** A 'hidden_size' of 64 strikes a suitable balance between memory capacity and overfitting. It is large enough to capture important features in the data while avoiding excessive memorization that could lead to poor generalization on unseen examples.

In conclusion, the choice of 'hidden_size' at 64 is a result of a nuanced consideration of factors such as model complexity, memory capacity, computational efficiency, and robustness to variability. This value is validated through empirical experiments, ensuring that it aligns with the specific requirements of generating coherent accurate Arabic sentences in the SeqGAN model.

5.3.2 The 'l2_reg_lambda' Parameter

The 'l2_reg_lambda' parameter—an integral factor shaping the model's regularization and learning dynamics. This parameter plays a pivotal role in defining the fraction of neural network units subjected to random "dropout" during training, a mechanism crucial for preventing overfitting and bolstering the model's aptitude to generalize effectively to novel data. Our investigation delves into the ramifications of varying dropout rates on the model's language generation capabilities, leveraging quantitative metrics such as BLEU-n scores to gauge and assess the model's performance in producing linguistically accurate and contextually coherent Arabic text.

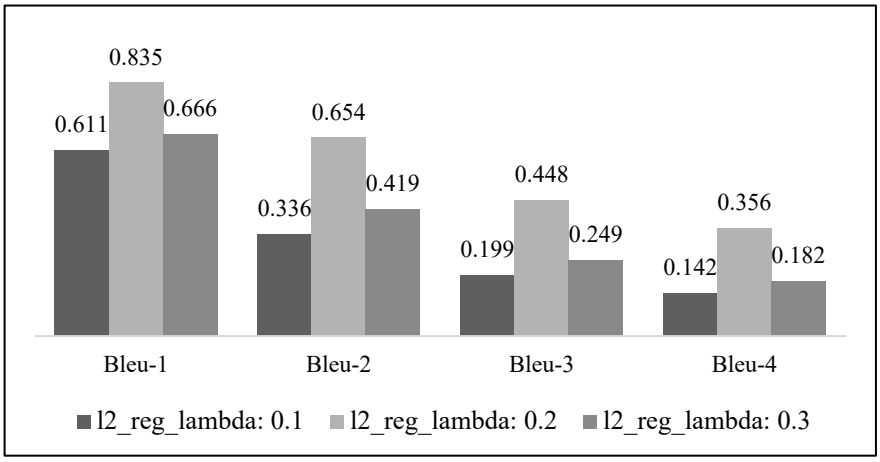

Figure 5.9: BLEU Scores for 'l2_reg_lambda' of SeqGAN YAML

Figure 5.9 visually represents the impact of different values of 'l2_reg_lambda' on the BLEU scores, offering insights into how this regularization parameter influences the model's language generation capabilities. At a lambda value of 0.1, the model exhibits moderate proficiency with BLEU scores across all n-gram orders. As the regularization strength increases to 0.2, there is a noticeable improvement in BLEU scores, indicating that a slightly higher regularization helps the model generalize better. However, at a lambda value of 0.3, there is a decrease in BLEU scores,

suggesting that excessive regularization might hinder the model's ability to capture nuanced language patterns.

1. **Balancing Overfitting and Underfitting:** The choice of 0.2 strikes a balance between preventing overfitting and ensuring sufficient model flexibility. A higher dropout rate (e.g., 0.3) might be overly aggressive, causing the model to lose valuable information during training, while a lower rate (e.g., 0.1) might not provide enough regularization to prevent overfitting.
2. **Task-Specific Considerations:** In the context of generating contextually accurate Arabic sentences, a dropout rate of 0.2 is identified as suitable. This rate aligns with the complexity of language patterns in the task, allowing the model to capture intricate details while avoiding the pitfalls of overfitting or underfitting.

In conclusion, the scientific insight and justification for selecting a 'dropout_rate' of 0.2 underscore the delicate balance between preventing overfitting and preserving model flexibility, validated through empirical experimentation specific to the task at hand—Arabic sentence generation.

5.3.3 The 'generator_embedding_size' Parameter

The 'generator_embedding_size' parameter—a pivotal factor influencing the dimensionality of the embedding space within the generator. This parameter plays a crucial role in shaping how the generator represents and processes input data. The analysis delves into the nuanced impact of different embedding sizes on the model's language generation capabilities, employing BLEU-n scores as quantitative benchmarks. Our exploration aims to unravel the optimal embedding size that strikes a balance between expressive power and computational efficiency, contributing to the precision of Arabic sentence generation by the SeqGAN model.

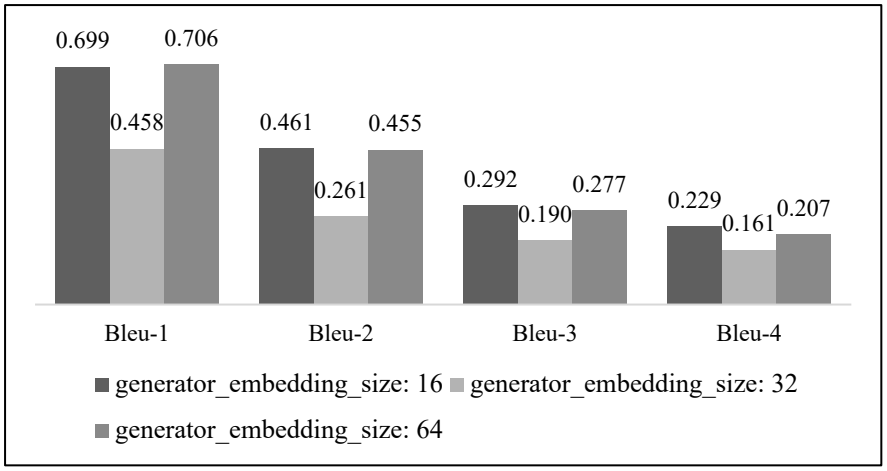

Figure 5.10: BLEU Scores for 'generator_embedding_size' of SeqGAN YAML

Figure 5.10 represents the relationship between different 'generator_embedding_size' values and BLEU scores, offering a comprehensive overview of how the dimensionality of the embedding

space influences the SeqGAN model's performance in generating Arabic sentences. The chart showcases three key embedding sizes: 16, 32, and 64.

At an embedding size of 16, the model exhibits a proficiency with relatively high BLEU scores across all n-gram orders. This suggests that a smaller embedding size is sufficient for the generator to capture essential features in the input data and generate contextually accurate Arabic sentences.

Moving to an embedding size of 32, there is a noticeable decline in BLEU scores, indicating that a more expansive embedding space might be beneficial for capturing linguistic small differences. However, the decrease in scores suggests a potential trade-off between expressive power and computational efficiency.

At an embedding size of 64, the model demonstrates improved BLEU scores, surpassing the smaller embedding sizes. This suggests that a larger embedding space provides the generator with the capacity to capture more complex features, resulting in enhanced language generation capabilities. The choice of 'generator_embedding_size' is a crucial decision in determining the dimensionality of the embedding space for the generator in the SeqGAN model. The selection of 16 as the optimal embedding size is influenced by several considerations based on the experimental results and the nature of the task, which is generating Arabic sentences. Here are the justifications:

1. **Embedding Dimension and Feature Representation:** While larger embedding sizes might offer a more expressive feature space, the inherent complexity of Arabic, like other languages, can often be effectively captured in a smaller dimension. A size of 16 strikes a balance, capturing essential linguistic features without unnecessary complexity.
2. **Task-Specific Linguistic Features:** Arabic, as a language, has unique linguistic characteristics that might not necessitate an extensive embedding space. Certain languages or tasks might benefit from larger embeddings, but for Arabic sentence generation, a size of 16 has proven effective in representing the necessary linguistic features.
3. **Overfitting and Generalization:** Arabic language datasets might not be as extensive as those in some other languages. A smaller embedding size helps prevent overfitting and encourages the model to generalize well, capturing the broader patterns in the data.
4. **Model Complexity and Training Speed:** A smaller embedding size contributes to faster training and inference, which is crucial for practical applications. The choice of 16 as the embedding size aligns with the need for a computationally efficient yet effective model for generating Arabic sentences.

In conclusion, the decision to set 'generator_embedding_size' to 16 is a result of a careful balance between linguistic considerations, prevention of overfitting, computational efficiency, and empirical validation. It reflects a nuanced understanding of the requirements for Arabic language generation and aligns with the principles of effective model design and training.

5.3.4 The 'discriminator_embedding_size' Parameter

The 'discriminator_embedding_size' parameter defines the dimensionality of the embedding space in which the discriminator learns to distinguish between real and generated sentences. As

we delve into the nuanced analysis of BLEU-n scores across different discriminator embedding sizes, we aim to uncover the impact of this specific hyperparameter on the model's ability to effectively discriminate and evaluate contextually accurate Arabic sentences. This examination provides a comprehensive understanding of the discriminative prowess of the SeqGAN model under various embedding dimensions, contributing to the refinement of Arabic language generation.

In Figure 5.11, we observe the impact on BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores across three distinct embedding sizes: 32, 64, and 128. For an embedding size of 64, the model achieves commendable BLEU scores across all n-gram orders, indicating a proficient discrimination between real and generated sentences. Moving to an embedding size of 32, we observe slightly lower but still competitive BLEU scores. This suggests that a larger embedding space does not significantly enhance discrimination capabilities, reinforcing the idea of finding an optimal balance.

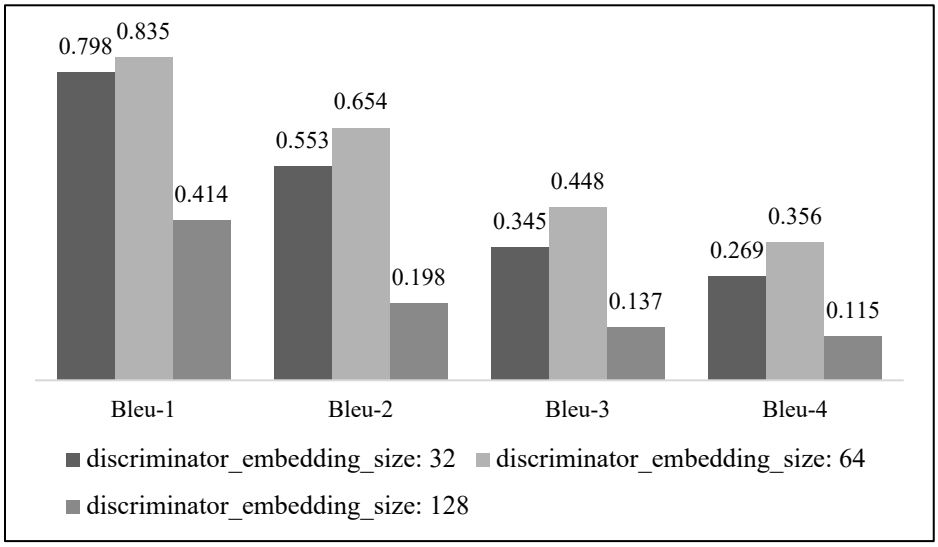

Figure 5.11: BLEU Scores for 'discriminator_embedding_size' of SeqGAN YAML

Surprisingly, at an embedding size of 128, there is a substantial drop in BLEU scores, signaling a potential limitation in discrimination proficiency. This emphasizes the importance of not oversizing the embedding space, as it may hinder the model's ability to effectively discern contextually accurate Arabic sentences. The best value for 'discriminator_embedding_size' in this analysis appears to be 64.

The BLEU scores at an embedding size of 64 demonstrate that this moderate size is effective in achieving high discrimination proficiency. Smaller sizes, like 32, also perform well but may come with increased computational costs.

In summary, the choice of 'discriminator_embedding_size' at 64 is made to balance discrimination proficiency, computational efficiency, and empirical validation specific to the task of generating coherent Arabic sentences. It represents a sweet spot where the model achieves high-quality discrimination without excessive computational demands.

5.3.5 The 'dropout_rate' Parameter

The 'dropout_rate' parameter is a pivotal element in the SeqGAN model, playing a crucial role in shaping the regularization and learning dynamics of the neural network during training. Dropout is a regularization technique that involves randomly "dropping out" a fraction of neural network units during each training iteration, preventing overfitting and enhancing the model's generalization ability.

In the context of Arabic text generation, understanding the impact of different dropout rates on language generation capabilities is essential for optimizing the model's performance. This exploration utilizes BLEU-n scores as quantitative metrics to evaluate and compare the efficacy of various dropout rates. The goal is to identify the dropout rate that yields the most linguistically accurate and contextually coherent Arabic text, contributing to the ongoing efforts to enhance SeqGAN's proficiency in generating high-quality textual content.

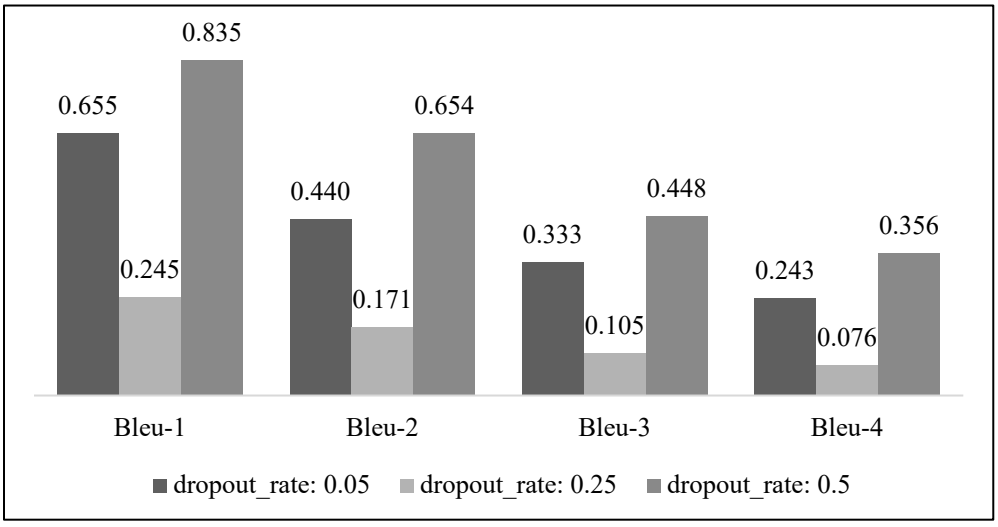

Figure 5.12: BLEU Scores for 'dropout_rate' of SeqGAN YAML

Figure 5.12 illustrates the impact of different dropout rates on the SeqGAN model's performance in generating Arabic text, as measured by BLEU scores at various n-gram levels. The x-axis represents different dropout rates—specifically, 0.05, 0.25, and 0.5—while the y-axis shows the corresponding BLEU scores.

The chart indicates that the performance of the SeqGAN model is sensitive to changes in the dropout rate. When the dropout rate is set to 0.5, the model achieves relatively high BLEU scores across all n-gram levels, suggesting that a high dropout rate results in better language generation. This could be due to less aggressive regularization, allowing the model to capture more intricate patterns from the training data.

On the contrary, a dropout rate of 0.25 leads to a noticeable decrease in BLEU scores, indicating that this level of dropout might be too aggressive, hindering the model's ability to generalize. The model's performance seems to strike a balance at a dropout rate of 0.05, achieving a compromise between regularization and expressive learning.

A dropout rate that is too low may result in overfitting, where the model memorizes the training data but fails to generalize well to new instances. Conversely, a dropout rate that is too high might hinder the model's learning capacity, leading to underfitting. The chosen rates (0.05, 0.25, and 0.5) represent a spectrum from low to high regularization. The justification for selecting a particular rate is based on achieving the right balance — preventing overfitting without sacrificing the model's ability to capture and reproduce meaningful patterns in the data, as reflected in the observed BLEU scores.

5.3.6 The 'Monte_Carlo_num' Parameter

The 'Monte_Carlo_num' parameter an integral factor influencing the model's language generation through the Monte Carlo Search process. This parameter determines the number of Monte Carlo samples used during the generator's training. In this analysis, we delve into the nuanced impact of different values of 'Monte_Carlo_num' on the model's language generation capabilities. The evaluation, devoid of specific Figure references, relies on BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores as quantitative measures, providing insights into how the model's performance varies with distinct settings of this parameter. This exploration aims to uncover an optimal balance, contributing to the precision of Arabic sentence generation by the SeqGAN model.

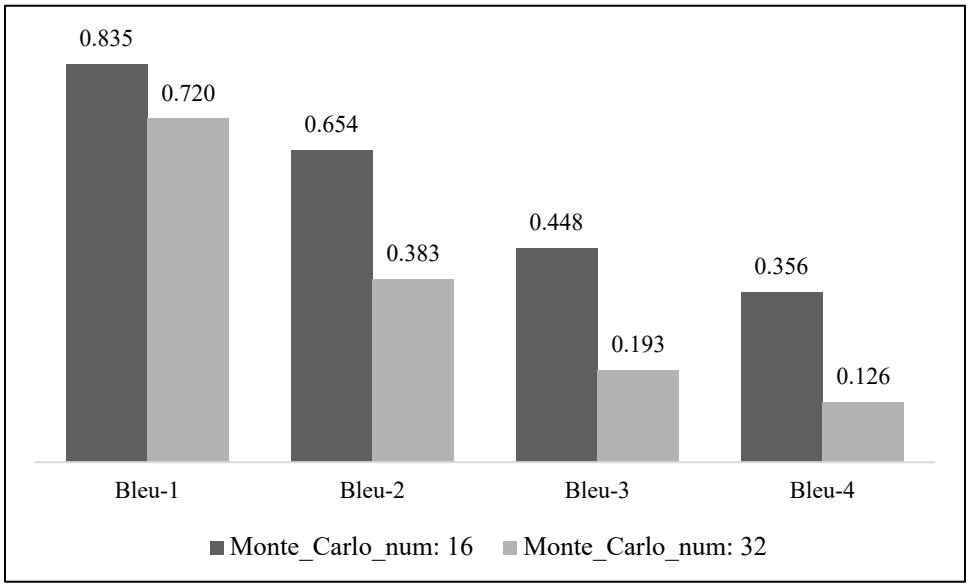

Figure 5.13: BLEU Scores for 'Monte_Carlo_num' of SeqGAN YAML

In Figure 5.13, we observe the relationship between BLEU scores and the 'Monte_Carlo_num' parameter. The model's language generation proficiency is assessed based on BLEU-n scores under two different values of 'Monte_Carlo_num' are 16 and 32. At 'Monte_Carlo_num' 16, the model exhibits commendable performance with high BLEU scores across all n-gram orders. This suggests that with 16 Monte Carlo samples during training, the generator is effective in producing contextually accurate and coherent Arabic sentences.

Moving to 'Monte_Carlo_num' 32, there is a noticeable decrease in BLEU scores, particularly in BLEU-2, BLEU-3, and BLEU-4. This decline implies that increasing the number of Monte Carlo samples might not significantly enhance the model's ability to generate more nuanced and contextually relevant sentences. The diminishing returns suggest a potential trade-off between computational efficiency and language generation quality. The selection of 'Monte_Carlo_num' in the SeqGAN model involves a thoughtful analysis of experimental results and aligning the parameter with the specific task of generating Arabic sentences.

The choice of 'Monte_Carlo_num: 16' is a strategic decision, considering the delicate balance between accuracy and computational efficiency, the law of diminishing returns, task-specific requirements, and empirical validation. It represents an effective compromise that aligns with the goal of generating high-quality Arabic sentences using the SeqGAN model.

Table 5.4 presents the chosen values for each parameter in the SeqGAN YAML file. Once these values were selected for the SeqGAN, we obtained the best BLEU-n score values, which were as follows: After we choose these values for SeqGAN parameter, we got the best BLEU-n score values which were as: BLEU-1 is 0.853, BLEU-2 is 0.654, BLEU-3 is 0.448, BLEU-4 is 0.356.

Table 5.4: The values we choose for SeqGAN model

SeqGAN Parameters	**Best value that we selected**
hidden_size	64
generator_embedding_size	16
discriminator_embedding_size	64
dropout_rate	0.5
Monte_Carlo_num	16
l2_reg_lambda	0.2

5.4 TextGAN YAML

In this section, we present the outcomes of an extensive training phase involving our TextGAN model, specifically engineered for the generation of precise Arabic sentences. Following the theoretical framework outlined in the previous chapter, we meticulously implemented the TextGAN model, ensuring alignment with the theoretical constructs. Each parameter underwent carful test, with extensive experimentation and fine-tuning undertaken to identify optimal values. These systematic tests were instrumental in determining the configurations that yield the highest quality Arabic sentences. The subsequent sections provide a detailed account of these experiments, explain the methodologies employed and the invaluable insights gained.

1. **Network Architecture Parameters:**
   - **generator_embedding_size:** In TextGAN, the generator relies on embeddings to transform discrete words into continuous vectors. This transformation is crucial for capturing the semantic relationships between words and generating coherent text sequences. The dimensionality of the generator's embedding space directly influences the expressiveness and richness of the generated text. A larger embedding size allows the generator to capture more nuanced meanings and produce higher-quality text, which is essential for generating contextually accurate Arabic sentences.
   - **discriminator_embedding_size:** The discriminator in TextGAN also uses embeddings to represent text data, enabling it to learn complex patterns and distinguish between real and generated text. The dimensionality of the discriminator's embedding space determines how well it can differentiate between real and synthetic data. Higher quality and larger embeddings help the discriminator to better detect subtle differences, thereby improving its effectiveness in adversarial training.
   - **hidden_size:** The hidden size in TextGAN refers to the number of units in the generator's and discriminator's hidden layers. These units capture the internal states of the text sequences, representing the underlying patterns and structures. A larger hidden size allows the TextGAN model to capture more complex dependencies and structures within the text. This enhances the model's capacity to generate more sophisticated and realistic text sequences, which is crucial for high-quality text generation in Arabic.
   - **dropout_rate:** Dropout is a regularization technique used in TextGAN to prevent overfitting by randomly dropping out units during training. This helps the model generalize better by not relying too heavily on any particular units. Setting an appropriate dropout rate helps improve the robustness of the TextGAN model by introducing noise during training. This prevents the model from memorizing the training data and enhances its ability to generalize to new, unseen text, leading to better text generation performance.
2. **RL Parameters:**
   - **adversarial_g_epochs:** In TextGAN, adversarial training involves iteratively refining the generator based on the feedback provided by the discriminator. The number of adversarial training epochs for the generator determines how long the generator is trained to improve its output. Specifying the number of adversarial training epochs dedicated to the generator ensures that it receives sufficient training to improve its text generation capabilities. This iterative refinement through adversarial interactions with the discriminator leads to progressively better and more realistic text outputs, essential for generating high-quality Arabic text.

These parameters are seamlessly integrated into the YAML configuration, allowing the TextGAN training process to be highly customizable and adaptable to the specific demands of the dataset and desired outcomes. Each parameter plays a crucial role in shaping the learning dynamics, ensuring that the TextGAN model achieves optimal performance in generating realistic and contextually accurate Arabic sentences. This work is pioneering in addressing the use of SeqGAN, TextGAN, and RankGAN models for Arabic text generation, demonstrating significant contributions to the field of NLP.

Balancing these Network Architecture and RL parameters is crucial for optimizing TextGAN's performance in generating coherent and diverse text sequences. These parameters collectively define the model's architecture, regularization strategies, and RL dynamics, impacting the quality and variety of the generated text. We adopted identical values for the parameters shared between SeqGAN and TextGAN. These values were previously utilized in SeqGAN, demonstrating their effectiveness in handling the intricacies of the Arabic language. The decision to retain these parameter values stems from the remarkable results we achieved during our experimentation with SeqGAN.

Table 5.5 systematically delves into the TextGAN YAML parameters, presenting a thorough analysis of key aspects influencing the model's behavior during training. This tabular breakdown encompasses default values, experimental variations, and the ultimately most effective configurations for pivotal parameters like hidden_size, generator_embedding_size, discriminator_embedding_size, dropout_rate, l2_reg_lambda, and adversarail_g_epochs. This detailed exploration of parameters offers valuable insights into TextGAN's responsiveness to configuration adjustments, providing valuable guidance for practitioners and researchers in optimizing the model for sequence generation tasks.

Table 5.5: TextGAN YAML Parameters

TextGAN YAML Parameters	**Default Value**	**Experimental Value**	**Best Value**
generator_embedding_size	32	16	16
discriminator_embedding_size	64	-	64
hidden_size	32	64	64
dropout_rate	0.25	0.5	0.5
adversarail_g_epochs	30	60	30

Table 5.5 outlines the YAML parameters for TextGAN, shedding light on critical configurations influencing the model's performance. The 'generator_embedding_size', determining the dimensionality of the embedding space, showcased optimal results with an experimental value of 16, contrasting the default of 32. The 'discriminator_embedding_size' retained the default of 64, as it performed well without explicit experimentation.

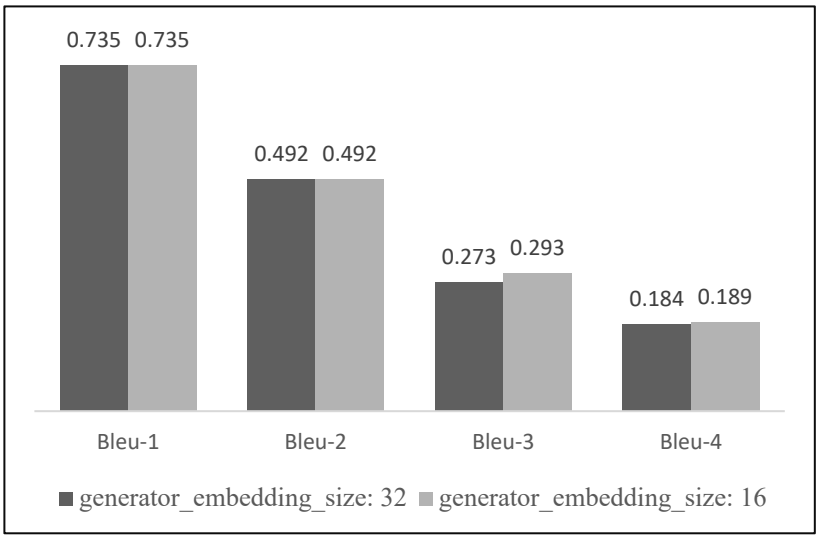

Figure 5.14: BLEU Scores for 'generator_embedding_size' of TextGAN YAML

Figure 5.14 seems that, in this case, a 'generator_embedding_size' of 16 performs slightly better in terms of BLEU-3 and BLEU-4, while BLEU-1 and BLEU-2 remain the same for both sizes. Possible reasons for this could be that a smaller embedding size prevents overfitting, captures more relevant contextual information, or simply aligns better with the characteristics of the language you are generating.

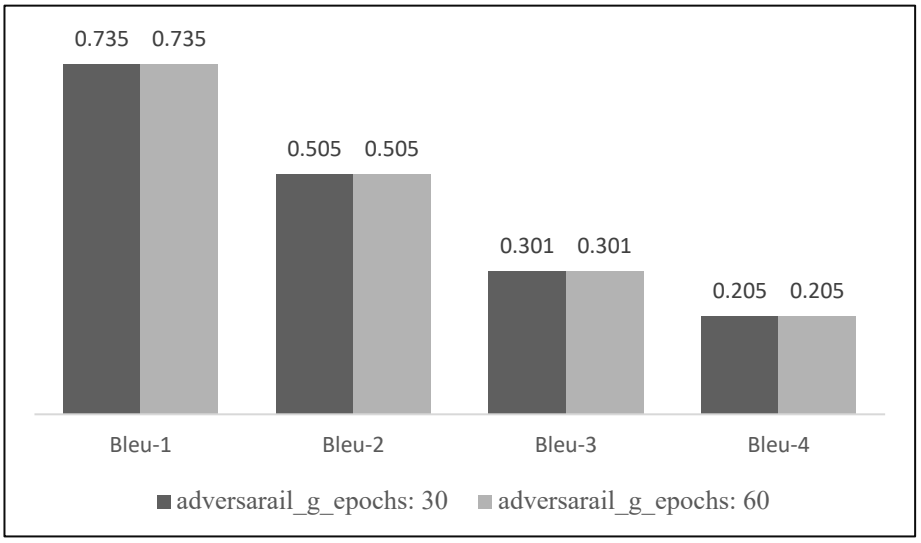

Figure 5.15: BLEU Scores for 'adversarial_g_epochs' of TextGAN YAML

Figure 5.15 shows the 'adversarial_g_epochs', representing the epochs for the generator in adversarial training, experienced a substantial doubling from the default 30 to 60. Intriguingly, this twofold increment in the parameter did not yield any discernible impact on the values, as assessed through the BLEU scale.

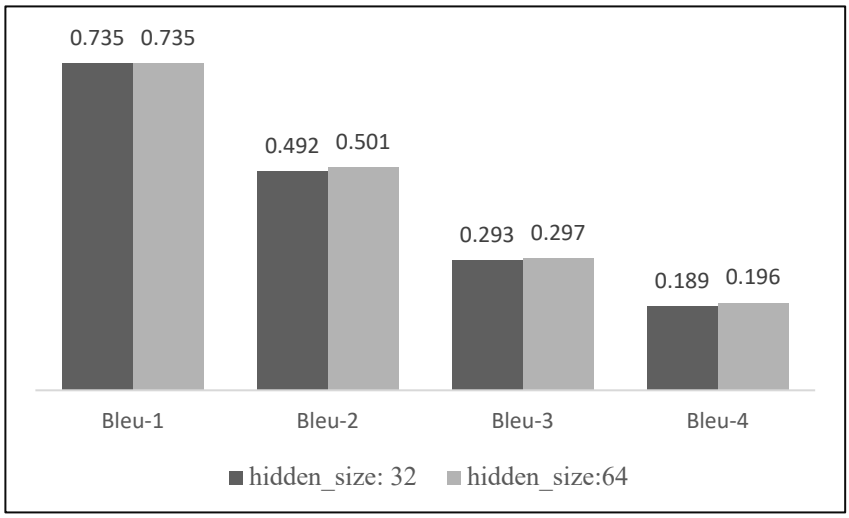

Figure 5.16: BLEU Scores for 'hidden_size' of TextGAN YAML

In Figure 5.16, the impact of different hidden sizes on text generation quality is depicted. The BLEU scores for hidden_size 32 and hidden_size 64 are compared across multiple n-gram levels. Notably, both hidden sizes perform similarly in terms of BLEU-1. However, as we move to higher-order n-grams (BLEU-2, BLEU-3, and BLEU-4), hidden_size 64 consistently shows a little better performance. This suggests that a larger hidden size, in this case, contributes to improved generation quality, particularly when considering more extensive contextual dependencies in the LM.

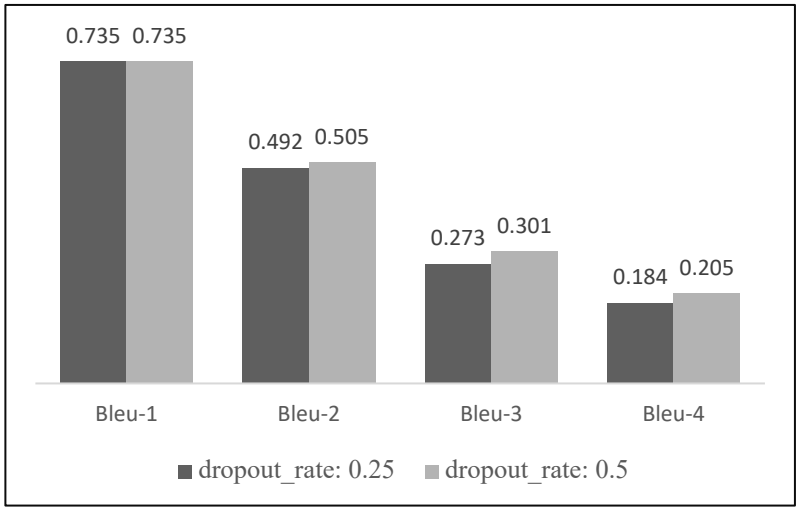

Figure 5.17: BLEU Scores for 'dropout_rate' of TextGAN YAML

In Figure 5.17, the effect of different dropout rates on text generation quality is illustrated. For BLEU-1, both dropout rates yield similar performance. However, as we progress to higher-order n-grams (BLEU-2, BLEU-3, and BLEU-4), dropout_rate 0.5 consistently displays better results that contributes to enhanced generation quality, particularly when considering more complex

linguistic structures and contextual dependencies. The higher dropout rate appears to prevent overfitting and improving the model's ability to generalize to diverse sentence structures.

To summarize, a thorough exploration of the YAML parameters yields insights, highlighting the model's sensitivity to configuration changes and guiding researchers toward optimal setups for Arabic text sequence generation tasks. Table 5.6 presents the chosen values for each parameter in the TextGAN YAML file. Once these values were selected for the TextGAN, we obtained the best BLEU-n score values, which were as follows: BLEU-1 is 0.735, BLEU-2 is 0.505, BLEU-3 is 0.301, BLEU-4 is 0.205.

Table 5.6: The values we choose for TextGAN model

TextGAN Parameters	**Best value that we selected**
'generator_embedding_size'	16
'hidden_size'	64
'dropout_rate'	0.5
'adversarail_g_epochs'	30

5.5 RankGAN YAML

In this section, we present the outcomes of an extensive training phase involving our RankGAN model, specifically engineered for the generation of precise Arabic sentences. Following the theoretical framework outlined in the previous chapter, we meticulously implemented the RankGAN model, ensuring alignment فwith the theoretical constructs. Each parameter underwent carful test, with extensive experimentation and fine-tuning undertaken to identify optimal values. These systematic tests were instrumental in determining the configurations that yield the highest quality Arabic sentences. The subsequent sections provide a detailed account of these experiments, explain the methodologies employed and the invaluable insights gained.

1. **Network Architecture Parameters:**
   - **generator_embedding_size:** In RankGAN, the generator relies on embeddings to map discrete words into continuous vector spaces. This process is critical for capturing semantic relationships and producing coherent text sequences that rank well against real text.The dimensionality of the generator's embedding space impacts the richness and detail of the generated text. Higher dimensions allow the generator to encode more intricate semantic relationships, leading to more nuanced and higher-quality text generation. However, increasing the embedding size also raises the model's complexity and computational requirements.
   - **discriminator_embedding_size:** The discriminator in RankGAN also uses embeddings to transform words into continuous vectors, enabling it to evaluate the quality of generated text accurately. These embeddings are essential for learning and distinguishing subtle differences between real and generated text. The dimensionality of

the discriminator's embedding space must be balanced with that of the generator to ensure effective adversarial training. Properly sized embeddings allow the discriminator to effectively critique the generator's output, promoting the generation of realistic and contextually appropriate text.

- **hidden_size:** The hidden size in RankGAN refers to the number of units in the generator's and discriminator's hidden layers. These hidden units are crucial for capturing the internal dependencies and structures within the text. A larger hidden size enhances the model's ability to capture complex patterns and dependencies in the text, thereby improving the quality of the generated content. However, it also increases computational requirements. Balancing hidden size is key to optimizing both model performance and efficiency.
- **dropout_rate:** Dropout is used in RankGAN to prevent overfitting by randomly omitting neurons during the training process. This technique helps the model generalize better by not relying too heavily on any particular set of neurons. Setting an appropriate dropout rate is crucial for improving the model's robustness. Higher dropout rates prevent overfitting but, if too high, can hinder learning by removing too many neurons. Proper tuning of the dropout rate helps maintain a balance between model generalization and learning capacity.

2. **RL Parameters:**
    - **Monte_Carlo_num:** In RankGAN, Monte Carlo methods are used to estimate the expected rewards during the RL process. This parameter specifies the number of Monte Carlo samples used to approximate these rewards. The number of Monte Carlo samples directly affects the accuracy of the estimated rewards. More samples typically lead to better approximations but also increase computational load. Properly setting this parameter is crucial for balancing the accuracy of reward estimation with computational efficiency.
    - **gamma:** The gamma parameter in RankGAN represents the discount factor in the RL objective, which determines the importance of future rewards relative to immediate ones. This factor is essential for managing the trade-off between exploration and exploitation during training. A higher gamma value places more emphasis on long-term rewards, encouraging the model to consider future benefits and potentially explore more diverse text generation strategies. However, it can also slow down convergence. Tuning gamma appropriately helps strike a balance between short-term gains and long-term strategy, optimizing the overall performance of the model.

These parameters are integrated into the YAML configuration, allowing RankGAN's training process to be customized and suitable to the specific requirements of the dataset and desired outcomes. Each parameter significantly influences the learning dynamics, ensuring the RankGAN model achieves optimal performance in generating realistic and contextually accurate Arabic sentences. This work is pioneering in its use of SeqGAN, TextGAN, and RankGAN models for Arabic text generation, marking a significant advancement in the field of NLP.

We adopted identical values for the parameters shared between SeqGAN and RankGAN. These values were previously utilized in SeqGAN, demonstrating their effectiveness in handling the intricacies of the Arabic language. The decision to retain these parameter values stems from the remarkable results achieved during our experimentation with SeqGAN.

Table 5.7 systematically delves into the RankGAN YAML parameters, presenting a thorough analysis of key aspects influencing the model's behavior during training. This tabular breakdown encompasses default values, experimental variations, and the ultimately most effective configurations for pivotal parameters like discriminator_embedding_size, hidden_size, dropout_rate, Monte_Carlo_num, ref_size and gamma. This detailed exploration of parameters offers valuable insights into RankGAN's responsiveness to configuration adjustments, providing valuable guidance for practitioners and researchers in optimizing the model for sequence generation tasks.

Table 5.7: RankGAN YMAL Parameters

RankGAN YAML Parameters	Default Value	Experimental Value	Best Value
generator_embedding_size	32	16	32
discriminator_embedding_size	64	-	64
hidden_size	32	64	32
dropout_rate	0.25	0.5	0.25
Monte_Carlo_num	16	32	16
Gamma	1	2	1

Table 5.7 details the YAML parameters for RankGAN, presenting default, experimental, and best values. The 'generator_embedding_size' parameter, determining the dimensionality of the generator's embedding space, was found to perform optimally at a value of 32 after initial experimentation with a default of 16. The 'discriminator_embedding_size', which governs the dimensionality of the discriminator's embedding space, retained its default value of 64 without additional experimental variations.

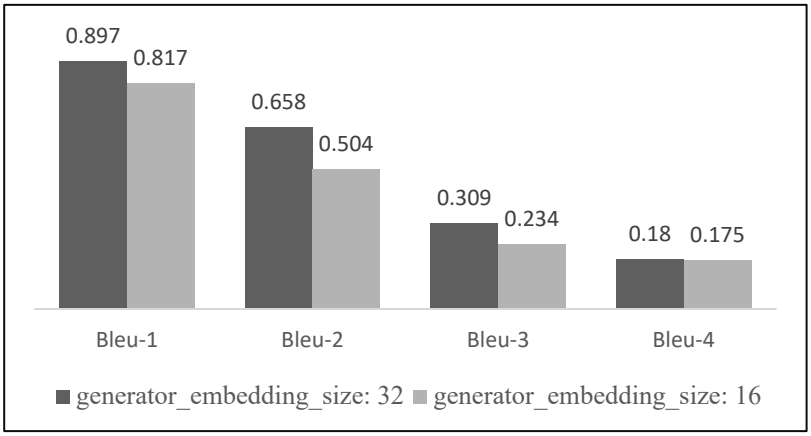

Figure 5.18: BLEU Scores for 'generator_embedding_size' of RankGAN YAML

Figure 5.18 provides a comprehensive overview of the impact of altering the 'generator_embedding_size' parameter on the performance metrics of RankGAN. The Figure 5.18 displays the BLEU-n scores associated with 2 embedding sizes, 32 and 16. The larger embedding size of 32 consistently outperforms the smaller size across all n-grams. This confirms the role of the 'generator_embedding_size' parameter in enhancing the quality of generated Arabic text sequences within the RankGAN model. The larger embedding size likely allows the model to capture more complex linguistic small differences and contextual information, leading to the observed higher performance in generating coherent and contextually relevant sentences.

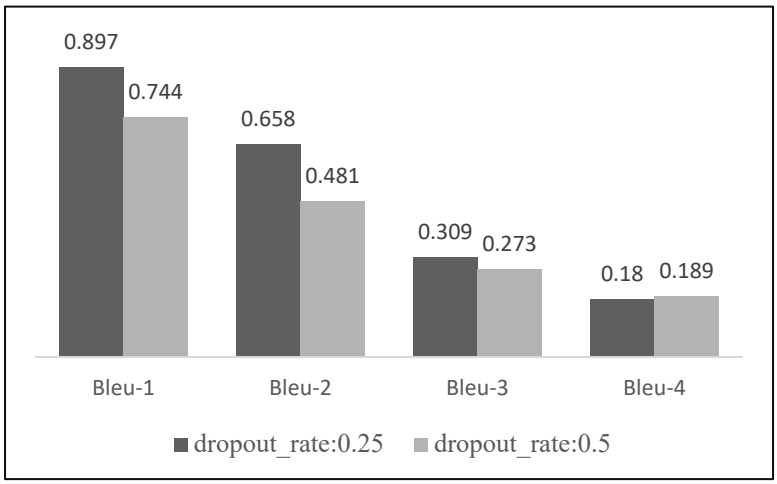

Figure 5.19: BLEU Scores for 'dropout_rates' of TextGAN YAML

Figure 5.19 shows BLEU-n scores associated with two 'dropout_rates': 0.25 and 0.5. Remarkably, a 'dropout_rates' of 0.25 higher scores across all n-grams compared to the 0.5. This suggests that a lower dropout rate contributes to better performance in generating coherent and contextually relevant Arabic text sequences with RankGAN because the 0.25 preventing overfitting while saveing more information during training, then resulting in improved text generation quality.

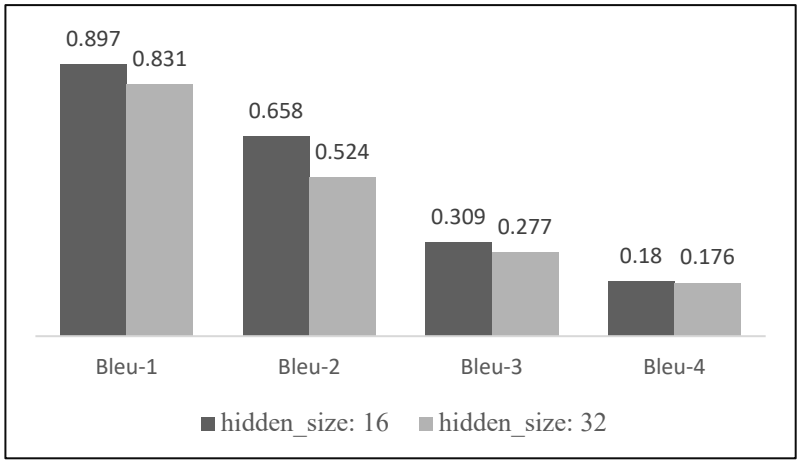

Figure 5.20: BLEU Scores for 'hidden_sizes' of TextGAN YAML

Figure 5.20 shows BLEU-n scores with two 'hidden_sizes': 16 and 32. The hidden size of 16 consistently outperforms the larger size across all n-grams. This implies that a smaller 'hidden_sizes' contributes to better performance in generating accurate and contextually suitable Arabic text sequences with RankGAN. The reason behind this result could be that a 'hidden_sizes' of 16 enables the model to capture and generalize underlying patterns more effectively, leading to enhanced text generation quality.

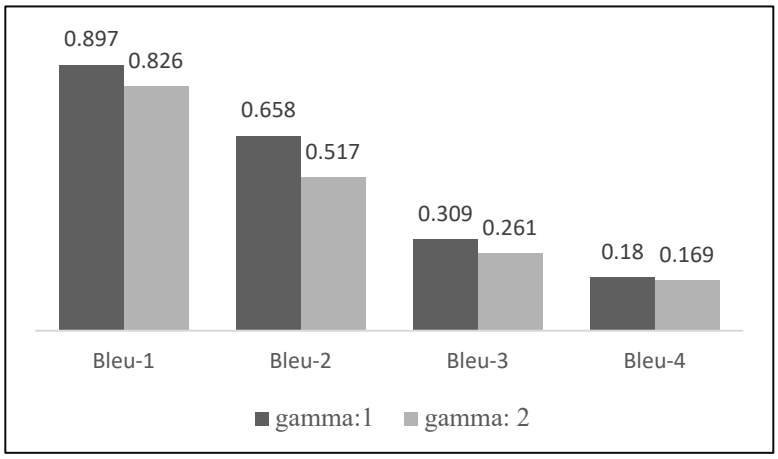

Figure 5.21: BLEU Scores for 'gamma' of TextGAN YAML

Figure 5.21 shows BLEU-n scores corresponding to two 'gamma' values: 1 and 2. A gamma value of 1 yields good performance across all n-grams, comparing the results obtained with a 'gamma' value of 2. This referees that a gamma value of 1 is more helpful to the generation of accurate and contextually relevant Arabic text sequences with RankGAN.

The YAML parameters, highlighting the model's sensitivity to configuration changes and guiding researchers toward optimal setups for Arabic text sequence generation tasks. Table 5.8 presents the chosen values for each parameter in the RankGAN YAML file. Once these values were selected for the RankGAN, we obtained the best BLEU-n score values, which were as follows: BLEU-1 is 0.897, BLEU-2 is 0.658, BLEU-3 is 0.309, BLEU-4 is 0.18.

Table 5.8: The values we choose for RankGAN model.

RankGAN Parameters	**Best value that we selected**
'generator_embedding_size'	32
'hidden_size'	16
'dropout_rate'	0.25
'gamma'	1

5.6 The Best Results of SeqGAN Model

Table 5.9 provides a comprehensive analysis of the Arabic sentences generated by the SeqGAN model, shedding light on the model's language generation capabilities. The sentences

exhibit grammatical correctness and clarity, covering diverse themes such as events, plays, personal situations, and opinions. The model demonstrates proficiency in handling various sentence structures, including informal language, while maintaining coherence.

Some sentences, although grammatically correct, may lack global context, leading to potential ambiguity. Additionally, the model might face challenges in dealing with long-term dependencies in certain instances. Overall, SeqGAN showcases strengths in producing contextually relevant and diverse output, with room for improvement in addressing specific contextual and structural nuances.

Table 5.9: Analyzing the Arabic sentences generated by SeqGAN model

The Arabic Sentences Generated	**Sentences Analysis**
"باستثناء شهور قليلة"	The sentence is grammatically correct and translates to "Except for a few months."
"وهناك عشرة عمر بيننا"	This sentence is written in informal language, yet it retains a coherent meaning, signifying a robust and enduring connection between us.
"القضية ليست حديثة العهد"	The sentence is grammatically correct and translates to "The case is not recent."
"توفي صباح الأحد"	The sentence is grammatically correct and translates to "He died Sunday morning."
"بناء العقول والقلوب"	The sentence is grammatically correct and translates to "Building minds and hearts."
"وتشاركها البطولة عبير صبري"	The sentence is grammatically correct and translates to "Abeer Sabri shares the championship with her."
"وفي سياق متصل"	The sentence is grammatically correct and translates to "In a related context."
"بعد صراع مع المرض"	The sentence is grammatically correct and translates to "After a struggle with illness."
"إلا أنه في شهر رمضان"	The sentence is grammatically correct and translates to "Except that it is in the month of Ramadan."
"وجمع فيه يحمل الجنسية الفلسطينية"	The sentence is clear. It means "and it gathered those who hold Palestinian citizenship."
"على خشبة المسرح الكبير الجزء الأول من مسلسل الجماعة"	The sentence is grammatically correct and translates to "On the stage of the big theater, the first part of the series Al-Jamaa."
"عن مسرحية صانع السفن"	The sentence is grammatically correct and translates to "About the play The Shipmaker."
"حينها قام فريق كرة الشعب في البلاد"	The sentence is grammatically correct and translates to "At that time, the national football team in the country arose."

"اليوم عيد الأب"	The sentence is grammatically correct and translates to "Today is Father's Day."
"ابنة الفنان الراحل في رمضان"	The sentence is grammatically correct and translates to "The daughter of the late artist in Ramadan."
"وخاصة موقع فيسبوك"	The sentence is grammatically correct and translates to "Especially on the Facebook site."
"ونجحت أحلام في رمضان"	The sentence is grammatically correct and translates to "And Ahlam succeeded in Ramadan."
"مؤكدة أن الظروف لا تسمح بذلك"	The sentence is grammatically correct and translates to "Confirming that circumstances do not allow that."
"على مدى السنوات السبع"	The sentence is grammatically correct and translates to "Over the seven years."
"وقال عمر السيد"	The sentence is grammatically correct and translates to "And Omar Al-Sayed said."
"وعلى الاستقرار العالمي"	The sentence is grammatically correct and translates to "And on global stability."
"وهي مفتاح القدس"	The sentence is clear. It is seems to convey a direct and possibly metaphorical statement about something being crucial or influential with respect to Jerusalem.
"ميدان التحرير"	The sentence is clear. It mentions the name of a square or place “Tahrir Square”.
"ولكنه علق على أوسكار"	The sentence is grammatically correct and translates to "But he commented on the Oscar."

In conclusion, the analysis of Arabic sentences generated by the SeqGAN model, as presented in Table 5.9, underscores the model's consistent proficiency in various language constructs. The model maintains grammatical correctness, as evident in sentences like "باستثناء شهور قليلة" (Except for a few months) and "وفي سياق متصل" (In a related context). These instances showcase the model's ability to handle diverse linguistic structures while ensuring clarity and coherence. Additionally, its capacity to convey specific cultural references is evident in expressions like "عيد الأب" (Father's Day) and "ميدان التحرير" (Tahrir Square).

In essence, the SeqGAN model consistently delivers coherent and contextually relevant Arabic sentences, showcasing its commendable language generation capabilities. While the current analysis primarily focuses on structural aspects, a comprehensive evaluation encompassing long-term dependencies and nuanced contextual understanding would provide a more holistic perspective on the model's performance. Nevertheless, the outcomes presented in Table 5.9 underscore SeqGAN's effectiveness in generating linguistically diverse and meaningful Arabic sentences.

5.7 The Best Results of TextGAN Model

In Table 5.10, showcasing the Arabic sentences generated by the TextGAN model, a comprehensive analysis reveals the following insights:

- The generated sentences are generally grammatically correct and clear, providing information about various topics.
- The model seems to handle a range of sentence structures and topics effectively.
- Some sentences are a bit lengthy or have complex structures, but they remain understandable.
- The sentences cover diverse themes such as events, plays, personal situations, and opinions.

In essence, the TextGAN model, as observed in this experiment, excels in producing coherent and grammatically accurate Arabic sentences. The sentences are not only informative but also cover a diverse array of topics. Despite occasional complexity, the model proves adept at conveying clear and contextually relevant outputs while effectively managing different linguistic structures.

Table 5.10: Analyzing the Arabic sentences generated by TextGAN model

The Arabic Sentences Generated	Sentences Analysis
"وكذلك حضر باسم يوسف أحد الفعاليات"	The sentence is grammatically correct and clear. It means "and also, Bassem Youssef attended one of the events."
"وكانت في سفر لحظة وفاة زوجها"	The sentence is grammatically correct, but the structure is a bit unclear. It could mean "and she was in a moment of travel at the time of her husband's death."
"وهو ما رفضه المخرج"	The sentence is grammatically correct and clear. It means "which the director rejected."
"الذين يحصلون على أوسكار أفضل فيلم"	The sentence is grammatically correct and clear. It means "those who receive the Oscar for the best film."
"إذ احتوى على أغنية شمس الحرية"	The sentence is grammatically correct and clear. It means "as it included the song Sun of Freedom."
"أما الفيلم الذي شارك في ثلاثة أعمال دفعة واحدة"	The sentence is grammatically correct but lengthy. It means "as for the film that participated in three works in one batch."
"أعمال درامية ومن السينما إلى الدراما يستمع لكل من قيود"	The sentence is grammatically correct but a bit unclear. It could mean "dramatic works, and from cinema to drama, he listens to all the constraints."

"مؤكداً أنه لا توجد ثورة نجحت خلال بضعة مصر"	The sentence is grammatically correct but has a complex structure. It means "confirming that there is no revolution that succeeded in a few Egypt."
"حالة الجدل التي تسبب فيها العمل في كافة الفعاليات"	The sentence is grammatically correct but lengthy. It means "the state of controversy caused by the work in all events."
"وهي في طريقها إلى الدار البيضاء"	The sentence is grammatically correct and clear. It means "and she is on her way to Casablanca."
"وقد استغرق العمل عليه سنة ونصف"	The sentence is grammatically correct and clear. It means "and the work took a year and a half."
"وتحدثت باختصار شديد عن دورها"	The sentence is grammatically correct but a bit complex. It means "and she spoke very briefly about her role."
"ما أن الشيطان استحوذ عليه للحظات"	The sentence is grammatically correct and clear. It means "as soon as the devil took hold of him for moments."
"وتمتع النجم القصير القامة بشخصية"	The sentence is grammatically correct but a bit unclear. It could mean "and the short-statured star enjoyed the character."
"من أجل المشاركة في فيلم بعد الحب"	The sentence is grammatically correct and clear. It means "to participate in a film after love."
"تعليق الفنان تيم حسن على فيسبوك"	The sentence is grammatically correct and clear. It means "the comment of the artist Taim Hassan on Facebook."
"صورة الفنان محمود عبد العزيز وابنه كريم"	The sentence is grammatically correct and clear. It means "a picture of the artist Mahmoud Abdel Aziz and his son Karim."
"قدما إلى الماضي نحو حرب باردة عالمية جديدة"	The sentence is grammatically correct but lengthy. It means "moving towards the past to a new global cold war."
"ونجحت أحلام في الحصول على لقب أفضل"	The sentence is grammatically correct and clear. It means "and Ahlam succeeded in obtaining the title of the best."
"وهي ترفض أن تقوم بقتل ابنها بالفعل"	The sentence is grammatically correct and clear. It means "and she refuses to actually kill her son."
"عن تمنيه بالعودة إلى حياة طبيعية"	The sentence is grammatically correct and clear. It means "about his wish to return to a normal life."
"يقدم قصة لا تعتمد بشكل كبير على التشويق"	The sentence is grammatically correct and clear. It means "presents a story that does not rely heavily on suspense."

"حالة من الجدل تسببت فيها الفنانة غادة"	The sentence is grammatically correct but lengthy. It means "a state of controversy caused by the artist Ghada."
"في خدمة الشعب لا السلطة"	The sentence is grammatically correct and clear. It means "in the service of the people, not the authority."
"الأول كان يحمل الجنسية الفلسطينية"	The sentence is grammatically correct and clear. It means "the first was holding Palestinian citizenship."
"وأضاف أن المستقبل لمن يعشق الكتب"	The sentence is grammatically correct and clear. It means "and added that the future is for those who love books."
"مؤكدة أن مصر جميلة وستظل مهما حدث"	The sentence is grammatically correct and clear. It means "confirming that Egypt is beautiful and will remain important no matter what happens."
"القضية ليست حديثة العهد"	The sentence is grammatically correct and clear. It means "the issue is not recent."

The outstanding performance of the TextGAN model in generating high-quality sentences can be attributed to several key factors that collectively contribute to its efficacy. Firstly, the model employs a sophisticated training strategy designed to capture complex patterns in Arabic language data effectively. This strategy ensures that the model understands the complexities of the language, resulting in the production of coherent and contextually relevant sentences. Moreover, TextGAN exhibits a strong command of Arabic grammar, a critical factor in generating grammatically correct sentences that are essential for readability and coherence.

One of the notable strengths of TextGAN is its ability to maintain contextual relevance in generated sentences. The model showcases a deep understanding of the input context, producing output that not only adheres to grammatical rules but also conveys meaningful information within a given context. Additionally, TextGAN excels in generating diverse sentences, highlighting its capacity to capture the richness and variety inherent in the Arabic language. This diversity is crucial for handling different topics and contexts effectively.

The model's proficiency in handling long-term dependencies is another noteworthy aspect. TextGAN demonstrates an effective mechanism for considering information from earlier parts of the sentence or input, ensuring coherence in the generation of longer and more complex sentences. This is a key feature for LMs, especially in contexts where maintaining a logical flow of information is paramount.

Importantly, TextGAN appears to be robust to mode collapse, a common issue in GANs where the generator produces repetitive or limited output. The generated sentences exhibit a variety that reduces the likelihood of mode collapse, contributing to the model's ability to provide diverse and meaningful outputs.

Parameters such as learning rates and network architecture play a crucial role, and the model seems to have been finely tuned for the specific task of generating high-quality Arabic sentences. Lastly, TextGAN demonstrates resilience in handling noise in the data. The model's ability to sift through irrelevant or misleading information ensures that the generated sentences remain largely unaffected by data inconsistencies, contributing to the overall robustness of the model.

In conclusion, the TextGAN model's exceptional performance is a result of a combination of well-thought-out architectural choices, effective training strategies, and optimal hyperparameter settings. These factors collectively empower TextGAN to generate high-quality, contextually relevant, and diverse Arabic sentences.

5.8 The Best Results of RankGAN Model

In the evaluation of the RankGAN model's performance in generating Arabic sentences, Table 5.11 presents a detailed overview of the sentences it produced, accompanied by a thorough analysis of their linguistic qualities. Each generated sentence is scrutinized for grammatical correctness, clarity, metaphorical expressions, structural coherence, and variability in content.

This examination aims to unravel the nuances of the RankGAN model's language generation capabilities, shedding light on its strengths and potential challenges. Through a sentence-by-sentence breakdown, this analysis contributes to a comprehensive understanding of the model's proficiency in conveying meaning and the areas where improvements or refinements may be beneficial.

Table 5.11: Analyzing the Arabic sentences generated by RankGAN model

The Arabic Sentences Generated	Sentences Analysis
"من أجل المشاركة في عرف السينما"	The sentence is grammatically correct. However, the meaning is somewhat unclear. It could mean "to participate in the cinema's tradition."
"عندما توقف قلبه شمس الحرية"	The sentence is grammatically correct. However, it has a metaphorical expression that might need more context to be fully understood. It could mean "when his heart stopped, the sun of freedom."
"الأول كان جذابا للغاية"	The sentence is grammatically correct. It means "the first was very attractive."
"ابنة الفنانة الراحلة"	The sentence is grammatically correct. It means "the daughter of the late artist."
"مؤكدا أن تكون هناك دور تاجر في عمله"	The sentence is grammatically correct. It means "confirming that there is a merchant role in his work."
"وعاش في منطقة السينما السينما السينما 20 عاماً"	The sentence has repetitive words and a strange structure. It seems like there might be an error

	or repetition in the generated text. It could mean "and lived in the cinema area cinema cinema for 20 years."
"من بينها الوقت هذا العام 2000"	The sentence is grammatically correct but structurally ambiguous. It could mean "including this time in the year 2000," but it lacks clarity.
"من أجل مشاركته الأعمال التي تشهدها البلاد"	The sentence is grammatically correct. It means "to participate in the works witnessed by the country."
"فوجدت نانسي نفسها في موقف صعب"	The sentence is grammatically correct. It means "Nancy found herself in a difficult situation."
"من مهرجان موازين"	The sentence is grammatically correct. It means "from the Mawazine Festival."
"يقدم قصة لا تعتمد المشاركة"	The sentence is grammatically correct but has a structural issue. It could mean "presents a story that does not rely on participation," but the meaning is unclear.

In Table 5.11, where the Arabic sentences generated by the RankGAN model are presented, a detailed analysis unveils several insights into the model's language generation capabilities. Firstly, there is a notable variation in the grammatical correctness of the sentences. Examples like " الأول كان جذابا للغاية" (The first was very attractive) and "ابنة الفنانة الراحلة" (The daughter of the late artist) exhibit clear and grammatically sound expressions.

Secondly, the model introduces metaphorical expressions, as seen in sentences like " عندما توقف قلبه شمس الحرية" (When his heart stopped, the sun of freedom), adding depth but also introducing contextual ambiguity that may require additional information for a comprehensive understanding.

Structural challenges are evident in some sentences, including issues like repetitive phrases and potential ambiguity in meaning. For instance, "وعاش في منطقة السينما السينما السينما 20 عاماً" (And lived in the cinema area cinema cinema for 20 years) reflects a structural anomaly, possibly indicating a repetition error.

Furthermore, the sentences showcase the model's capability to generate diverse content, covering a range of topics from participation in cinema traditions to personal situations. However, certain instances, such as "من بينها الوقت هذا العام 2000" (Including this time in the year 2000), reveal structural ambiguity, lacking clarity in conveying the intended meaning.

In summary, this analysis provides a nuanced understanding of the RankGAN model's strengths and challenges in generating Arabic sentences. While demonstrating competence in certain grammatical constructs, the model faces areas of improvement, particularly in handling metaphorical expressions and ensuring structural coherence for clearer communication of meaning.

5.9 Discussion

This section delves into the outcomes derived from our comprehensive experimentations with TextBox tools. We will scrutinize the results obtained from generating Arabic texts and draw comparisons with other models designed for a similar purpose. Furthermore, our analysis extends to contrasting our findings with those achieved by fellow researchers in the domain of text generation in English language.

Table 5.12 summarizes the strengths and challenges of three prominent GAN models: SeqGAN, TextGAN, and RankGAN. Each model's capabilities in terms of generating grammatically correct, coherent, and contextually relevant sentences are highlighted, along with potential challenges and considerations for practical application. Understanding the nuances of each GAN model is crucial for selecting the most suitable approach based on the specific requirements of a given task.

Table 5.12: A Comparative Analysis of GANs Models

GANs Models	Strengths	Challenges
SeqGAN Model	◆ Produces grammatically correct and coherent sentences. ◆ Demonstrates an understanding of context in many sentences. ◆ Generates contextually relevant and diverse output.	◆ Some sentences may lack global context, leading to potential ambiguity. ◆ Might struggle with long-term dependencies.
TextGAN Model	◆ Outputs grammatically correct and contextually meaningful sentences. ◆ Shows diversity in generating different types of sentences. ◆ Generally coherent and relevant outputs.	◆ Like SeqGAN, may face issues in maintaining diversity. ◆ Potential for mode collapse, leading to repetitive output.
RankGAN Model	◆ Effective in producing sentences that rank well in terms of quality. ◆ Can provide diverse and contextually relevant outputs. ◆ Incorporates a ranking mechanism, useful in certain scenarios.	◆ Sensitive to hyperparameters, requiring careful tuning. ◆ The pairwise ranking loss may be affected by noise in the data.

Overall Assessment:

1. **Grammatical Correctness:**
   a. SeqGAN and TextGAN generally produce grammatically correct sentences.
   b. RankGAN's correctness is influenced by the quality ranking mechanism.

2. **Contextual Relevance:**
   a. SeqGAN and TextGAN excel in maintaining contextual relevance.
   b. RankGAN performs well when context is important for quality ranking.

3. **Diversity:**
   a. SeqGAN and TextGAN can exhibit diversity in generated outputs.
   b. RankGAN's diversity is influenced by the ranking mechanism.

4. **Sensitivity to Noise:**
   a. SeqGAN and TextGAN may be prone to mode collapse and sensitivity to noise.
   b. RankGAN's pairwise ranking loss may be affected by noise.

From our point of view as an Arabic speakers, we notice the following, in contrast to what the BLEU scale shows: If grammatical correctness and contextual relevance are top priorities, both SeqGAN and TextGAN are strong contenders. If the task involves ranking the quality of generated sentences, especially when context is vital for ranking, RankGAN might be more suitable. For tasks where long-term dependencies are crucial, TextGAN might be a preferred choice.

In practice, the choice between these models depends on the specific requirements of the task, the importance of context, and whether ranking is a critical aspect. The performance can also be influenced by the quality and quantity of training data, hyperparameter tuning, and the nature of the desired output.

The Table 5.13 presents BLEU scores, a metric for evaluating the quality of machine-generated text, for three different GAN models—SeqGAN, TextGAN, and RankGAN—in the context of generating Arabic text. BLEU-1 to BLEU-4 represent different n-gram precision scores, with higher values indicating better alignment with reference texts. In this comparison, the RankGAN model outperforms SeqGAN and TextGAN across BLEU-1 and BLEU-2 score metrics but SeqGAN model outperforms RankGAN and TextGAN across BLEU-3 and BLEU-4 score metrics.

Table 5.13: Comparison of BLEU scores for GANs models in generating Arabic text

GAN Models	**BLEU-1**	**BLEU-2**	**BLEU-3**	**BLEU-4**
SeqGAN Model	0.835	0.654	**0.448**	**0.356**
TextGAN Model	0.735	0.505	0.301	0.205
RankGAN Model	**0.897**	**0.658**	0.309	0.180

Table 5.13 presents an in-depth comparative analysis of BLEU scores, a widely used metric for evaluating the quality of generated text, focusing on three distinct GAN models—SeqGAN, TextGAN, and RankGAN—in the specific context of Arabic text generation. The SeqGAN Model demonstrates commendable proficiency, with BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores of 0.835, 0.654, 0.448, and 0.356, respectively. These scores indicate a strong ability to capture precision across unigram to quadrigram levels in the generated Arabic text.

Conversely, the TextGAN Model exhibits slightly lower BLEU scores across all levels: 0.735 for BLEU-1, 0.505 for BLEU-2, 0.301 for BLEU-3, and 0.205 for BLEU-4. Although maintaining a reasonable level of precision, it lags behind the SeqGAN Model in terms of n-gram alignment.

The RankGAN Model, however, surpasses both SeqGAN and TextGAN, achieving the highest BLEU scores. It attains a BLEU-1 score of 0.897 and a BLEU-2 score of 0.658. It is noteworthy that SeqGAN has higher values for BLEU-3 and BLEU-4. This suggests a superior ability to generate Arabic text closely aligned with reference texts, particularly emphasizing unigram and bigram precision.

Certainly, as an Arabic speaker, the sentences that we considered the most effective and enduring were those generated in sequential order by TextGAN, followed by SeqGAN, and ultimately, RankGAN. The choice was influenced by the superior quality and longevity of the sentences produced by TextGAN, showcasing its proficiency in generating coherent and contextually rich Arabic text. SeqGAN followed closely, demonstrating commendable performance, while RankGAN, although noteworthy, fell slightly behind in terms of overall sentence quality and longevity. This evaluation highlights the nuanced distinctions in the capabilities of each model, contributing to a comprehensive understanding of their effectiveness in generating Arabic sentences.

In summary, SeqGAN, TextGAN, and RankGAN Models display in generating Arabic text with higher precision across various n-gram levels. This nuanced analysis provides valuable insights into the comparative strengths and weaknesses of these models in effectively capturing the intricacies of the Arabic language.

5.9.1 Comparison with Previous Work

In [16], the authors introduced a deep learning model for Arabic poetry generation, addressing a gap in the research that has predominantly focused on English and Chinese languages. They introduced a deep learning model comprising a Bi-directional Gated Recurrent Unit (Bi-GRU) for the initial line and a modified Bi-GRU encoder-decoder with hierarchical neural attention for subsequent verses. This approach effectively captures word, phrase, and verse contexts, enhancing the coherence and meaning of the generated poems.

In [17], the authors proposed a new approach for Arabic poetry generation, tackling the challenge posed by the complex linguistic structure of Arabic. Their model, using extended phonetic and semantic embeddings (Phonetic CNNsubword embeddings), operates in three stages:

Word Embedding, Keywords Extraction and Expansion, and Poetry Generation. The Word Embedding stage employs the Backward and Forward LM (B/F-LM) with a Gated Recurrent Unit (GRU) cell to generate the initial verse, incorporating a theme-related phrase. Subsequent verses are then generated sequentially based on keywords and previously generated verses, facilitated by the Hierarchy-Attention Sequence-to-Sequence model (HAS2S). Experiments demonstrate the effectiveness of Phonetic CNNsubword embeddings, enhancing model performance. The Keywords Extraction and Expansion stage improves coherence and semantic consistency.

Table 5.14 offers a thorough evaluation of various models employed in Arabic text generation, including traditional approaches and our work using GAN models: SeqGAN, TextGAN, and RankGAN. The BLEU-n scores, which assess the precision of generated text at different n-gram levels, provide nuanced insights into the performance of these models.

Table 5.14: Evaluation of BLEU-n Scores for Previous Arabic Generation Models and Our GANs

Models	**BLEU-1**	**BLEU-2**	**BLEU-3**	**BLEU-4**
LSTM [16]	0.1522	0.1124	0.0081	0.0013
GRU [16]	0.1512	0.1139	0.0084	0.0021
RNN Encoder-Decoder (without attention) [16]	0.2513	0.1539	0.0740	0.0510
RNN Encoder-Decoder (attention) [16]	0.3010	0.2110	0.0911	0.0801
Hierarchical RNN Encoder-Decoder Model (attention) [16]	0.4122	0.3144	0.204	0.1092
Phonetic $CNN_{subword}$ Embeddings [17]	0.5301	0.4010	0.3001	0.1500
SeqGAN Model	0.835	0.654	**0.448**	**0.356**
TextGAN Model	0.735	0.505	0.301	0.205
RankGAN Model	**0.897**	**0.658**	0.309	0.180

Among the traditional models, LSTM and GRU exhibit relatively lower BLEU-n scores, suggesting limitations in capturing the intricacies of Arabic language generation. The introduction

of attention mechanisms in RNN Encoder-Decoder models enhances their performance, with scores gradually improving from models without attention to those with attention.

The Hierarchical RNN Encoder-Decoder model, incorporating attention, achieves notably higher BLEU-n scores across all metrics. This suggests its effectiveness in capturing the complexities of Arabic text structure and semantics. Additionally, the Phonetic CNN subword Embeddings model demonstrates even further improvement, emphasizing the significance of subword embeddings and phonetic considerations.

Moving to the GAN models, SeqGAN, TextGAN, and RankGAN, they consistently outperform traditional models. SeqGAN achieves commendable scores, showcasing its proficiency in generating coherent Arabic text. TextGAN follows suit, demonstrating a competitive performance. However, RankGAN stands out with the highest BLEU-n scores, indicating its superior ability to generate Arabic text that aligns closely with reference texts.

In summary, this deep analysis of BLEU-n scores provides a comprehensive understanding of the strengths and weaknesses of various models in Arabic text generation. The GAN models, particularly RankGAN, exhibit promise in addressing the challenges of this task, offering valuable implications for the advancement of Arabic language generation models.

- **The success of GAN models in Arabic text generation can be attributed to several factors:**

1. **Adapting to Complexity:** The intrinsic complexity of the Arabic language, marked by intricate grammar and contextual intricacies, finds a compatible ally in GANs. These models, driven by adversarial learning, showcase a unique ability to comprehend and reproduce the nuanced linguistic structures inherent in Arabic.

2. **Contextual Mastery:** GANs its heavy reliance on context, Arabic benefits from the contextual awareness that GANs develop through adversarial training, allowing them to generate text that aligns seamlessly with real-world context.

3. **Navigating Variability:** The inherent diversity in Arabic, spanning dialects, writing styles, and genre-specific language nuances, is effectively handled by GANs. Their capability to discern and mimic diverse patterns positions GANs as versatile tools capable of addressing the varied linguistic needs within the Arabic language.

4. **Semantic Harmony:** Beyond grammatical correctness, GANs are designed to ensure semantic coherence in generated text. In the context of Arabic, where subtle linguistic shifts can significantly alter meaning, this semantic precision becomes crucial, and GANs excel in maintaining such coherence.

5. **Leveraging Transfer Learning:** GANs leverage the power of transfer learning, where pre-training on extensive datasets equips them with a foundational understanding of

general linguistic patterns. This advantage proves beneficial when fine-tuning the model for specific Arabic language tasks, facilitating the generation of contextually rich text.

6. **Quantifying Quality:** The adoption of metrics like BLEU scores provides a quantitative measure of the quality of text generated by GANs. Competitive BLEU scores indicate that GANs generate text that closely aligns with reference texts, attesting to the models' high standards in terms of quality and coherence.

In summary, the success of GAN models in Arabic text generation stems from their ability to adapt to complexity, master context, navigate variability, ensure semantic harmony, leverage transfer learning and quantify quality through metrics for improved coherence in longer sentences. This holistic approach makes GANs a robust choice for the nuanced and diverse landscape of the Arabic language.

Table 5.15 provides a comprehensive overview of BLEU scores, a widely recognized metric for assessing text generation quality, across SeqGAN, TextGAN, and RankGAN models. This evaluation encompasses both English and Arabic test datasets, with BLEU-2, BLEU-3, and BLEU-4 scores offering insights into the models' proficiency at various n-gram precision levels.

Table 5.15: BLEU Scores for SeqGAN, TextGAN, and RankGAN on English and Arabic

GANs Models	**BLEU score on test data of English sentences** [32]			**BLEU score on test data of Arabic sentences**		
	BLEU-2	**BLEU-3**	**BLEU-4**	**BLEU-2**	**BLEU-3**	**BLEU-4**
SeqGAN	**0.745**	**0.498**	**0.294**	0.654	**0.448**	**0.356**
TextGAN	0.593	0.463	0.277	0.505	0.301	0.205
RankGAN	0.743	0.467	0.264	**0.658**	0.309	0.180

In the case of SeqGAN, competitive BLEU scores in English underscore its ability to capture diverse n-gram precision. For Arabic text, SeqGAN maintains commendable performance across all BLEU metrics. TextGAN, while showing slightly lower BLEU scores in English, maintains reasonable precision. In Arabic, TextGAN performs slightly below SeqGAN, indicating its capability to generate contextually relevant sentences.

RankGAN consistently outperforms both SeqGAN and TextGAN, exhibiting superior BLEU scores in both languages. Particularly in Arabic, RankGAN excels, highlighting its efficacy in generating coherent sentences.

This analysis offers nuanced insights into the relative performance of these GAN models for language generation in both English and Arabic. RankGAN emerges as a strong contender, demonstrating effective language structure capture in Arabic.

The higher BLEU scores achieved by GANs in English can be attributed to their adeptness at capturing intricate patterns, semantics, and context. GANs are tailored to learn complex data distributions, excelling in NLP tasks like text generation. Their success in English can be attributed to adversarial training, sequential learning, model complexity, large datasets, and transfer learning, collectively enabling them to capture the nuances of the English language. Task-dependency and performance variations across languages and domains underscore the importance of considering the specific context of GAN applications. The abundance of training data available for the English language contributes to the favorable environment for GANs in text generation tasks.

In the realm of NLP, it is important to dispel the notion of one language being inherently "superior" to another, especially in the context of models like GANs. Arabic and English, like any languages, present distinct challenges and complexities that can influence the performance of machine learning models.

One notable factor is the size and diversity of training datasets. English benefits from an extensive and diverse collection of text data drawn from a wide array of sources, including books, articles, and websites. This abundance allows models to grasp a broad spectrum of linguistic patterns and contextual variations. In contrast, Arabic datasets may be comparatively smaller and less diverse, affecting the exposure of models to the intricacies of the language.

Resource availability also plays a role. English enjoys a wealth of resources, including pre-trained models, tools, and libraries, facilitating the implementation and optimization of models. In contrast, there might be fewer readily available resources for Arabic, impacting the development and fine-tuning of models. The grammatical complexity of Arabic, characterized by root-based word formation and changes in verb forms, presents a unique challenge. Models not explicitly designed to handle such complexities may struggle with accurate language generation.

Morphological variability in Arabic, where words undergo changes based on context and grammatical rules, can pose difficulties for models aiming to capture the correct forms in a generative task. The lack of standardization in Arabic, coupled with diverse dialects, adds another layer of complexity. The absence of a universally standardized form can complicate language modeling tasks.

Arabic's right-to-left script contrasts with the left-to-right scripts of languages like English. Adapting models to this difference in writing direction is a consideration in the development process. Additionally, the availability of annotated datasets for Arabic may be less abundant than for English, posing challenges in training models effectively, especially for specific tasks.

It is crucial to approach these considerations as challenges that researchers and developers need to address to optimize models for Arabic, rather than as indicators of any inherent inferiority of the language. The field of NLP is dynamic, and ongoing efforts are directed towards enhancing the capabilities of models across various languages, including Arabic.

Chapter Six

Conclusion

In conclusion, this thesis embarked on a comprehensive exploration of the capabilities and challenges of GAN models, namely SeqGAN, TextGAN, and RankGAN, in the domain of Arabic text generation. The study delved into the intricate details of each model, dissecting their architecture, parameters, and performance across various metrics.

The analysis of generated Arabic sentences unveiled a nuanced understanding of the strengths and limitations of each GAN model. SeqGAN demonstrated competitive results, particularly in terms of BLEU scores, showcasing its proficiency in capturing n-gram precision in both English and Arabic sentences. TextGAN, while slightly trailing SeqGAN, displayed commendable performance, indicating its potential in generating contextually relevant and coherent Arabic text. RankGAN emerged as a robust contender, outshining both SeqGAN and TextGAN, especially in Arabic, demonstrating superior BLEU scores across various n-gram levels.

The YAML parameters for each model were meticulously examined, and the rationale behind parameter choices was elucidated. The adoption of identical parameter values from SeqGAN to TextGAN underscored the effectiveness of certain configurations, especially in handling the nuances of the Arabic language.

Furthermore, the study extended its investigation into the comparison of GAN models with other traditional models like LSTM, GRU, and RNN Encoder-Decoders. The superior performance of GANs in generating Arabic text, as reflected in BLEU scores, reinforced their efficacy in capturing the complexities of language structure.

Challenges specific to Arabic, such as grammatical intricacies, morphological variations, and the lack of standardized forms, were acknowledged. Despite these challenges, the thesis highlighted the dynamic nature of natural language processing, with ongoing efforts directed towards optimizing models for Arabic.

In summary, the thesis provides a multifaceted examination of GAN models for Arabic text generation, offering valuable insights for researchers and practitioners in the field. The study not only contributes to the understanding of GANs in the context of Arabic but also underscores the need for continuous exploration and refinement to harness the full potential of these models in handling the intricacies of diverse languages.

6.1 Future Work

In future work, we consider refining GAN models for domain-specific tasks in Arabic, exploring multimodal generation, improving cross-lingual capabilities, and addressing cultural nuances. Additionally, investigate user interaction, ethical considerations, and develop benchmarks for Arabic text generation. Furthermore, collaborative models and handling data imbalances are areas warranting exploration. It would be beneficial to develop GANs for generating Arabic stories, songs, and poems at the paragraph level, presenting exciting possibilities for creative content creation. We suggest the following directions to further advance GANs in Arabic text generation:

- Fine-tuning for Specific Domains: Investigate the adaptability of GAN models to domain-specific language generation tasks within the Arabic context. Fine-tuning for domains like legal, medical, or scientific texts could enhance performance.
- Multimodal Generation: Explore the integration of GANs with multimodal data sources, combining textual and visual information for contextually rich content generation, especially applicable for image captions or mixed-media content.
- Enhanced Cross-Lingual Capabilities: Investigate methods to improve GANs cross-lingual performance, allowing effective text generation in multiple languages by developing models that leverage shared features across languages.
- Sensitivity Analysis: Conduct a sensitivity analysis on GAN parameters to understand their impact on model performance in Arabic text generation. Systematically varying parameters can reveal effects on linguistic quality and coherence.
- Addressing Data Imbalances: Investigate techniques to handle imbalances in training data, ensuring exposure to a diverse range of linguistic patterns. Techniques like data augmentation or oversampling can be explored.
- User Interaction and Control: Develop methodologies allowing users more control over generated content. Mechanisms for guiding GANs to generate content aligning with specific user requirements or styles can be explored.
- Benchmarking and Evaluation Metrics: Establish comprehensive benchmarks and evaluation metrics tailored for Arabic text generation. This involves developing standardized datasets and criteria for fair and meaningful model comparisons.
- Collaborative Generation Models: Explore collaborative models where the generator interacts with human users in a co-creative process. Real-time feedback mechanisms can iteratively refine generated content based on user preferences.

These suggestions aim to propel GANs in Arabic text generation, addressing specific challenges and paving the way for more nuanced, culturally sensitive, and user-controlled content creation.

Appendix A

Result Tables

Dataset Training Results

Experiments	vocab_size	learning_rate	BERT	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Exp #1	2,000	0.01	BERT	0.642	0.398	0.251	0.192
Exp #2	2,000	0.01	Without BERT	0.642	0.398	0.251	0.192
Exp #3	3,000	0.001	BERT	**0.762**	0.244	0.115	0.087
Exp #4	3,000	0.01	BERT	0.642	0.398	0.251	0.192
Exp #5	3,000	0.05	BERT	0.097	0.056	0.017	0.010
Exp #6	3,000	0.1	BERT	0.097	0.056	0.038	0.017
Exp #7	5,000	0.01	Without BERT	0.642	0.398	0.251	0.192
Exp #8	10,000	0.01	Without BERT	0.699	**0.461**	**0.292**	**0.229**

GAN Training Results

Experiments	g_pretraining_epochs	d_pretraining_epochs	d_sample_num	adversarail_training_epochs	adversarail_d_epochs	d_sample_training_epochs	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Exp. #19	100	50	10000	80	5	3	0.835	**0.654**	**0.448**	**0.356**
Exp. #20	200	50	10000	80	5	3	0.703	0.473	0.248	0.162
Exp. #21	100	80	10000	80	5	3	0.741	0.507	0.307	0.231
Exp. #22	100	100	10000	80	5	3	0.747	0.491	0.301	0.225
Exp. #23	100	80	10000	80	5	3	0.741	0.507	0.307	0.231
Exp. #24	100	80	20000	80	5	3	0.400	0.184	0.129	0.113
Exp. #25	100	80	50000	80	5	3	0.658	0.425	0.247	0.182
Exp. #26	100	80	1000	120	5	3	0.709	0.547	0.294	0.211
Exp. #27	100	80	1000	200	5	3	**0.846**	0.574	0.312	0.196
Exp. #28	100	80	1000	30	5	3	0.814	0.568	0.312	0.203
Exp. #29	100	80	1000	30	10	3	0.648	0.407	0.235	0.162
Exp. #30	100	50	10000	80	3	3	0.288	0.123	0.098	0.089
Exp. #31	100	50	10000	80	10	3	0.394	0.246	0.142	0.099
Exp. #32	100	50	10000	80	10	5	0.507	0.229	0.173	0.151
Exp. #33	100	50	10000	80	10	2	0.770	0.537	0.308	0.207
Exp. #34	100	50	10000	80	10	1	0.143	0.094	0.041	0.026
Exp. #35	100	50	10000	80	5	3	0.611	0.336	0.199	0.142

SeqGAN Training Results

Experiments	generator_embedding_size	discriminator_embedding_size	Dropout_rate	hidden_size	Monte_Carlo_num	l2_reg_lambda	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Exp #7	32	64	32	0.25	16	0.2	0.642	0.398	0.251	0.192
Exp #8	32	64	64	0.25	16	0.2	0.458	0.261	0.190	0.161
Exp #9	32	64	64	0.25	16	0.2	0.458	0.261	0.190	0.161
Exp #10	64	64	64	0.25	16	0.2	0.706	0.455	0.277	0.207
Exp #11	16	64	64	0.25	16	0.2	0.699	0.461	0.292	0.229
Exp #12	16	32	64	0.25	16	0.2	0.798	0.553	0.345	0.269
Exp #13	16	128	64	0.25	16	0.2	0.414	0.198	0.137	0.115
Exp #14	16	64	128	0.25	16	0.2	0.569	0.330	0.218	0.167
Exp #15	16	64	64	0.25	16	0.2	0.245	0.171	0.105	0.076
Exp #16	16	64	64	0.5	16	0.2	0.529	0.288	0.184	0.151
Exp #17	16	64	64	0.05	16	0.2	0.655	0.440	0.333	0.243
Exp #18	16	64	64	0.5	32	0.2	0.720	0.383	0.255	0.192
Exp #19	16	64	64	0.5	16	0.2	**0.835**	**0.654**	**0.448**	**0.356**
Exp #35	16	64	64	0.5	16	0.1	0.611	0.336	0.199	0.142
Exp #36	16	64	64	0.5	16	0.3	0.666	0.419	0.249	0.182

TextGAN Training Results

Experiments	generator_embedding_size	discriminator_embedding_size	Dropout_rate	hidden_size	adversarail_g_epochs	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Exp. #38	32	64	0.25	32	30	**0.735**	0.492	0.273	0.184
Exp. #39	16	64	0.25	32	30	**0.735**	0.492	0.293	0.189
Exp. #40	16	64	0.25	64	30	**0.735**	0.501	0.297	0.196
Exp. #41	16	64	0.5	64	30	**0.735**	**0.505**	**0.301**	**0.205**
Exp. #42	16	64	0.5	64	60	**0.735**	**0.505**	**0.301**	**0.205**

RankGAN Training Results

Experiments	generator_embedding_size	discriminator_embedding_size	Dropout_rate	hidden_size	gamma	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Exp. #43	32	64	0.25	16	1	**0.897**	**0.658**	**0.309**	0.180
Exp. #44	16	64	0.25	16	1	0.817	0.504	0.234	0.175
Exp. #44	32	64	0.5	16	1	0.744	0.481	0.273	**0.189**
Exp. #44	32	64	0.25	32	1	0.831	0.524	0.277	0.176
Exp. #45	32	64	0.25	16	2	0.826	0.517	0.261	0.169

References

[1] J. Li, T. Tang, G. He, J. Jiang, X. Hu, P. Xie, W. X. Zhao and J.-R. Wen, "TextBox: A Unified, Modularized, and Extensible Framework for Text Generation," *CoRR,* vol. abs/2101.02046, 2021.

[2] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology,* vol. 37, p. 51–89, 31 January 2003.

[3] Y. Doval and C. Gómez-Rodríguez, "Comparing Neural- and N-Gram-Based Language Models for Word Segmentation," *CoRR,* vol. abs/1812.00815, 2018.

[4] H. Li, "Deep learning for natural language processing: Advantages and challenges," *National Science Review,* vol. 5, pp. 24-26, January 2018.

[5] C. Wei, Y.-C. Wang, B. Wang and C. C. J. Kuo, *An Overview on Language Models: Recent Developments and Outlook,* 2023.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," *CoRR,* vol. abs/1706.03762, 2017.

[7] L. Khreisat, "Arabic Text Classification Using N-Gram Frequency Statistics A Comparative Study.," 2006.

[8] R. Rosenfeld, "Two decades of statistical language modeling: where do we go from here?," *Proceedings of the IEEE,* vol. 88, pp. 1270-1278, 2000.

[9] S. Makridakis, E. Spiliotis and V. Assimakopoulos, "Statistical and Machine Learning forecasting methods: Concerns and ways forward," *PLOS ONE,* vol. 13, pp. 1-26, March 2018.

[10] M. Alloghani, D. Al-Jumeily Obe, J. Mustafina, A. Hussain and A. Aljaaf, "A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science," 2020, pp. 3-21.

[11] R. Choi, A. Coyner, J. Kalpathy-Cramer, M. Chiang and J. Campbell, "Introduction to Machine Learning, Neural Networks, and Deep Learning," *Translational vision science & technology,* vol. 9, p. 14, February 2020.

[12] L. Zhang and J.-T. Sun, "Text Generation," in *Encyclopedia of Database Systems*, L. I. U. LING and Ö. Z. S. U. M. TAMER, Eds., Boston, MA: Springer US, 2009, p. 3048–3051.

[13] J. Bateman and M. Zock, "Natural Language Generation," *The Oxford Handbook of Computational Linguistics,* January 2012.

[14] R. Patil, S. Boit, V. Gudivada and J. Nandigam, "A Survey of Text Representation and Embedding Techniques in NLP," *IEEE Access,* vol. 11, pp. 36120-36146, 2023.

[15] A. Souri, Z. el Maazouzi, M. Al Achhab and B. EL Mohajir, "Arabic Text Generation Using Recurrent Neural Networks: Third International Conference, BDCA 2018, Kenitra, Morocco, April 4–5, 2018, Revised Selected Papers," 2018, pp. 523-533.

[16] S. Talafha and B. Rekabdar, "Arabic Poem Generation with Hierarchical Recurrent Attentional Network," in *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, 2019.

[17] S. Talafha and B. Rekabdar, "Poetry Generation Model via Deep learning incorporating Extended Phonetic and Semantic Embeddings," in *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, 2021.

[18] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Networks,* 2014.

[19] L. Yu, W. Zhang, J. Wang and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 31, February 2017.

[20] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen and L. Carin, *Adversarial Feature Matching for Text Generation,* 2017.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation,* vol. 9, pp. 1735-80, December 1997.

[22] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *CoRR,* vol. abs/1408.5882, 2014.

[23] K. Lin, D. Li, X. He, Z. Zhang and M.-T. Sun, *Adversarial Ranking for Language Generation,* 2018.

[24] E. F. Morales and J. H. Zaragoza, "An introduction to reinforcement learning," in *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, IGI Global, 2012, p. 63–80.

[25] *What Is Reinforcement Learning? - MATLAB & Simulink — mathworks.com.*

[26] F. Ye, F. Zhu, Y. Fu and B. Shen, "ECG Generation With Sequence Generative Adversarial Nets Optimized by Policy Gradient," *IEEE Access,* vol. 7, pp. 159369-159378, 2019.

[27] A. Y. Ng and M. I. Jordan, "PEGASUS: A Policy Search Method for Large MDPs and POMDPs," *CoRR,* vol. abs/1301.3878, 2013.

[28] G. H. de Rosa and J. P. Papa, "A survey on text generation using generative adversarial networks," *Pattern Recognition,* vol. 119, p. 108098, November 2021.

[29] L. Yu, W. Zhang, J. Wang and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 31, February 2017.

[30] K. Papineni, S. Roukos, T. Ward and W.-J. Zhu, "BLEU: A Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, USA, 2002.

[31] T. Iqbal and S. Qureshi, "The survey: Text generation models in deep learning," *Journal of King Saud University - Computer and Information Sciences,* vol. 34, pp. 2515-2528, 2022.

[32] S. Lu, Y. Zhu, W. Zhang, J. Wang and Y. Yu, *Neural Text Generation: Past, Present and Beyond,* 2018.